LNCS 5689

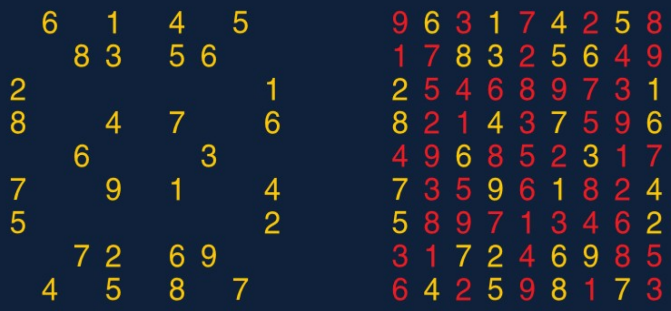
Tutorial

Sergio Tessaris Enrico Franconi
Thomas Eiter Claudio Gutierrez
Siegfried Handschuh Marie-Christine Rousset
Renate A. Schmidt (Eds.)

Reasoning Web

Semantic Technologies for Information Systems

5th International Summer School 2009
Brixen-Bressanone, Italy, August/September 2009
Tutorial Lectures



Springer

Sergio Tessaris   Enrico Franconi
Thomas Eiter   Claudio Gutierrez
Siegfried Handschuh
Marie-Christine Rousset   Renate A. Schmidt (Eds.)

Reasoning Web

Semantic Technologies for Information Systems

5th International Summer School 2009
Brixen-Bressanone, Italy, August 30 – September 4, 2009
Tutorial Lectures

Springer

Volume Editors

Sergio Tessaris
Enrico Franconi
Free University of Bozen - Bolzano, Italy
E-mail:{tessaris,franconi}@inf.unibz.it

Thomas Eiter
Technische Universität Wien, Austria
E-mail: eiter@kr.tuwien.ac.at

Claudio Gutierrez
Universidad de Chile, Chile
E-mail: cgutierr@dcc.uchile.cl

Siegfried Handschuh
National University of Ireland, Ireland
E-mail: siegfried.handschuh@deri.org

Marie-Christine Rousset
University of Grenoble, France
E-mail: Marie-Christine.Rousset@imag.fr

Renate A. Schmidt
The University of Manchester, UK
E-mail: schmidt@cs.man.ac.uk

Library of Congress Control Number: 2009932138

CR Subject Classification (1998): H.5, I.2.4, H.3, C.2, I.1, F.3, F.2

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN	0302-9743
ISBN-10	3-642-03753-4 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-03753-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12737505 06/3180 5 4 3 2 1 0

Preface

The Semantic Web is one of the major current endeavours of applied computer science. The Semantic Web aims at enriching the existing Web with meta-data and processing methods so as to provide Web-based systems with advanced (so-called intelligent) capabilities, in particular with context-awareness and decision support.

The advanced capabilities required in most Semantic Web application scenarios primarily call for reasoning. Reasoning capabilities are offered by Semantic Web languages that are currently being developed. Most of these languages, however, are developed mainly from functionality-centred perspectives (e.g., ontology reasoning or access validation) or application-centred perspectives (e.g., Web service retrieval and composition). A perspective centred on the reasoning techniques complementing the above-mentioned activities appears desirable for Semantic Web systems and applications. The Summer School is devoted to this perspective. The "Reasoning Web" series of annual Summer Schools was started in 2005 on behalf of the work package "Education and Training (ET)" of the Network of Excellence REWERSE.

This year's edition focused on the use of semantic technologies to enhance data access on the Web. For this reason, courses presented a range of techniques and formalisms which bridge semantic-based and data-intensive systems.

The school introduced Semantic Web foundations with a strong perspective on data management as well as applications of scalable semantic-based techniques for data querying. Topics of the lectures where design and analysis of reasoning procedures for Description Logics; Answer Set Programming basics, its modelling methodology and its principal extensions tailored for Semantic Web applications; languages for constraining and querying XML data; RDF databases theory and efficient and scalable support for RDF/OWL data storage, loading, inferencing and querying; tractable Description Logics and their use for Ontology-Based Data Access; the Social Semantic Desktop, which defines a user's personal information environment as a source and end-point of the Semantic Web.

We are grateful to all the lecturers and their co-authors for their excellent contributions, to the Reasoning Web School Board, and the organisations that supported this event: the Free University of Bozen–Bolzano and STI2 International.

August 2009

Thomas Eiter
Enrico Franconi
Claudio Gutierrez
Siegfried Handschuh
Marie-Christine Rousset
Renate Schmidt
Sergio Tessaris

School Organisation

Programme Chairs

Sergio Tessaris	Free University of Bozen-Bolzano, Italy
Enrico Franconi	Free University of Bozen-Bolzano, Italy

Programme Committee

Thomas Eiter	Vienna Technical University, Austria
Claudio Gutierrez	Universidad de Chile, Chile
Siegfried Handschuh	DERI Galway, Ireland
Marie-Christine Rousset	University of Grenoble, France
Renate Schmidt	University of Manchester, UK

Local Organisation

Stefano David	Università Politecnica delle Marche, Italy
Enrico Franconi	Free University of Bozen-Bolzano, Italy
Sergio Tessaris	Free University of Bozen-Bolzano, Italy

Sponsoring Institutions

Free University of Bozen-Bolzano
http://www.unibz.it

STI International
http://www.sti2.org

REWERSE
http://rewerse.net

External Reviewers

Pablo Barcelo	Universidad de Chile, Chile
Thomas Krennwallner	Vienna Technical University, Austria
Mantas Simkus	Vienna Technical University, Austria

Table of Contents

Description Logics

Franz Baader

Theoretical Computer Science, TU Dresden, Germany
baader@inf.tu-dresden.de

Abstract. Description Logics (DLs) are a well-investigated family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, and databases, but their most notable success so far is the adoption of the DL-based language OWL as standard ontology language for the semantic web.

This article concentrates on the problem of designing reasoning procedures for DLs. After a short introduction and a brief overview of the research in this area of the last 20 years, it will on the one hand present approaches for reasoning in expressive DLs, which are the foundation for reasoning in the Web ontology language OWL DL. On the other hand, it will consider tractable reasoning in the more light-weight DL $\mathcal{EL}$, which is employed in bio-medical ontologies, and which is the foundation for the OWL 2 profile OWL 2 EL.

1 Introduction

In their introduction to The Description Logic Handbook [11], Brachman and Nardi point out that the general goal of knowledge representation (KR) is to "develop formalisms for providing high-level descriptions of the world that can be effectively used to build intelligent applications" [32]. This sentence states in a compact way some of the key requirements that a KR formalism needs to satisfy. In order to be accepted as a *formalism* in this sense, a knowledge representation language needs to be equipped with a well-defined syntax and a formal, unambiguous semantics, which was not always true for early KR approaches such as semantic networks [101] and frames [90]. A *high-level* description concentrates on the representation of those aspects relevant for the application at hand while ignoring irrelevant details. In particular, this facilitates the use of relatively inexpressive languages even though they may not be able to faithfully represent the whole application domain. *Intelligent* applications should be able to reason about the knowledge and infer implicit knowledge from the explicitly represented knowledge, and thus the *effective* use of the knowledge depends on the availability of practical reasoning tools.

Description logics (DLs) [11] are a family of logic-based knowledge representation formalisms that are tailored towards representing the terminological knowledge of an application domain in a structured and formally well-understood way. They allow their users to define the important notions (classes, relations, objects) of the domain using concepts, roles, and individuals; to state constraints on the way these notions can be interpreted; and to deduce consequences such as subclass and instance relationships from the definitions and constraints. The name *description logics* is motivated by the fact that, on the one hand, classes are described by concept *descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL; on the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, *logic*-based semantics. For example, in a conference domain, we may have classes (concepts) like Person, Speaker, Author, Talk, Participant, PhD_student, Workshop, Tutorial; relations (roles) like gives, attends, attended_by, likes; and objects (individuals) like Richard, Frank, Paper_176. A speaker can be defined as a person that gives a talk:

$$\mathsf{Speaker} \equiv \mathsf{Person} \sqcap \exists \mathsf{gives}.\mathsf{Talk},$$

we can say that Frank is a speaker and attends the DL tutorial using the assertions:

$$\mathsf{Speaker}(\mathsf{FRANK}), \quad \mathsf{attends}(\mathsf{FRANK}, \mathsf{DL\_TUTORIAL}), \quad \mathsf{Tutorial}(\mathsf{DL\_TUTORIAL}),$$

and state the constraints that tutorials are only attended by PhD students:

$$\mathsf{Tutorial} \sqsubseteq \forall \mathsf{attended\_by}.\mathsf{PhD\_student},$$

and that the relation attended_by is the inverse of the relation attends:

$$\mathsf{attended\_by} \equiv \mathsf{attends}^{-1}.$$

DLs have been employed in various application domains, such as natural language processing, configuration, databases, and biomedical ontologies, but their most notable success so far is probably the adoption of the DL-based language OWL[1] as standard ontology language for the semantic web [69,15]. The three main reasons for the adoption of DLs as ontology languages are

– the availability of a formal, unambiguous semantics, which is based on the Tarski-style semantics of first-order predicate logic, and is thus fairly easy to describe and comprehend;
– the fact that DLs provide their users with various carefully chosen means of expressiveness for constructing concepts and roles, for further constraining their interpretations, and for instantiating concepts and roles with individuals;
– the fact that DL systems provide their users with highly-optimized inference procedures that allow them to deduce implicit knowledge from the explicitly represented knowledge.

[1] http://www.w3.org/TR/owl-features/

The formal semantics of DLs and typical concept and role constructors as well as the formalism for expressing constraints will be introduced in the next section. In the remainder of this section, we concentrate on the inference capabilities of DL systems. The *subsumption* algorithm determines subconcept-superconcept relationships: C is subsumed by D iff all instances of C are necessarily instances of D, i.e., the first concept is always interpreted as a subset of the second concept. For example, given the definition of $\mathsf{Speaker}$ from above, $\mathsf{Speaker}$ is obviously subsumed by Person. In general, however, induced subsumption relationships may be much harder to detect. The *instance* algorithm determines induced instance relationships: the individual i is an instance of the concept description C iff i is always interpreted as an element of C. For example, given the assertions for Frank and the DL tutorial from above, the constraint for tutorials, and the constraint expressing that attends is the inverse of attended by, we can deduce that FRANK is an instance of $\mathsf{Phd\_student}$. The *consistency* algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms, i.e., concept definitions and constraints) is non-contradictory. For example, if we added a disjointness constraint

$$\mathsf{Speaker} \sqcap \mathsf{PhD\_student} \sqsubseteq \bot$$

for speakers and PhD students to the conference knowledge base introduced so far, then this knowledge base would become inconsistent since it follows from the knowledge base that Frank is both a speaker and a PhD students, contradicting the stated disjointness of these two concepts.

In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable for the DL employed by the system, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the most important issues in DL research. The research related to this issue can be classified into the following five phases.[2]

Phase 1. (1980–1990) was mainly concerned with implementation of systems, such as KLONE, K-REP, BACK, and LOOM [33,88,100,87]. These systems employed so-called *structural subsumption algorithms*, which first normalize the concept descriptions, and then recursively compare the syntactic structure of the normalized descriptions [93]. These algorithms are usually quite efficient (polynomial), but they have the disadvantage that they are complete only for very inexpressive DLs, i.e., for more expressive DLs they cannot detect all the existing subsumption/instance relationships. At the end of this phase, early formal investigations into the complexity of reasoning in DLs showed that most DLs do not have polynomial-time inference problems [30,94]. As a reaction, the

[2] Note, however, that the assigned temporal intervals are only rough estimates, and thus should not be taken too seriously.

implementors of the Classic system (the first industrial-strength DL system) carefully restricted the expressive power of their DL [99,29].

Phase 2. (1990–1995) started with the introduction of a new algorithmic paradigm into DLs, so-called *tableau-based algorithms* [108,50,66]. They work on propositionally closed DLs (i.e., DLs with full Boolean operators) and are complete also for expressive DLs. To decide the consistency of a knowledge base, a tableau-based algorithm tries to construct a model of it by breaking down the concepts in the knowledge base, thus inferring new constraints on the elements of this model. The algorithm either stops because all attempts to build a model failed with obvious contradictions, or it stops with a "canonical" model. Since in propositionally closed DLs, subsumption and satisfiability can be reduced to consistency, a consistency algorithm can solve all inference problems mentioned above. The first systems employing such algorithms (Kris and Crack) demonstrated that optimized implementations of these algorithm lead to an acceptable behavior of the system, even though the worst-case complexity of the corresponding reasoning problems is no longer in polynomial time [14,35]. This phase also saw a thorough analysis of the complexity of reasoning in various DLs [50,51,49,47]. Another important observation was that DLs are very closely related to modal logics [103].

Phase 3. (1995–2000) is characterized by the development of inference procedures for very expressive DLs, either based on the tableau-approach [70,71] or on a translation into modal logics [44,45,43,46]. Highly optimized systems (FaCT, Race, and Dlp [67,61,98]) showed that tableau-based algorithms for expressive DLs lead to a good practical behavior of the system even on (some) large knowledge bases. In this phase, the relationship to modal logics [44,104] and to decidable fragments of first-order logic was also studied in more detail [28,96,59,57,58,75], and applications in databases (like schema reasoning, query optimization, and integration of databases) were investigated [36,40,42].

During *Phase 4* (2000–2005), industrial strength DL systems employing very expressive DLs and tableau-based algorithms were developed [115,62,109], with applications like the Semantic Web or knowledge representation and integration in bio-informatics in mind. In this phase, the Web Ontology Language OWL, whose sublanguages OWL DL and OWL Lite are based on expressive DLs, became an official W3C recommendation,[3] thus boosting the use of DLs for the definition of ontologies. On the more foundational side, this phase saw the development of alternative approaches for reasoning in expressive DLs, such as resolution-based approaches [73,74,2,72,78], which use an optimized translation of DLs into first-order predicate logic and then apply appropriate first-order resolution provers, and automata-based approaches [41,86,84,114,25,13], which are often more convenient for showing ExpTime complexity upper-bounds than tableau-based approaches.

We are now in *Phase 5*, where on the one hand even more expressive DLs with highly-optimized tableau-based algorithms [68] are proposed as basis for the

[3] http://www.w3.org/TR/owl-features/

new Web Ontology Language OWL 2.[4] On the other hand, more light-weight DLs are investigated and proposed as profiles of OWL 2,[5] such as members of the $\mathcal{EL}$ family [7,8], for which the subsumption and the instance problem are polynomial, and of the DL Lite family [38,39], for which the instance problem and query answering are polynomial w.r.t. data complexity. Another important development in this phase is that inference problems other than the classical ones (subsumption, instance, consistency) are gaining importance, such as query answering (i.e., answering conjunctive queries w.r.t. DL knowledge bases) [1,55,85,95], pinpointing (i.e., exhibiting the axioms responsible for a given consequence) [105,97,89,22,24], and modularization (i.e., extracting a part of a knowledge base that has the same consequence as the full knowledge base, for consequences formulated using a certain restricted vocabulary) [60,79,110].

2 Basic Definitions

As mentioned above, a key component of a DL is the *description language*, which allows its users to build complex concepts (and roles) out of atomic ones. These descriptions can then be uses in the *terminological part* of the knowledge base (TBox) to introduce the terminology of an application domain, by defining concepts and imposing additional (non definitional) constraints on their interpretation. In the *assertional part* of the knowledge base (ABox), facts about a specific application situation can be stated, by introducing named individuals and relating them to concepts and roles. *Reasoning* then allows us to derive implicit knowledge from the explicitly represented one. In the following, we introduce these four components of a DL more formally.

2.1 The Basic Description Language $\mathcal{ALC}$ and Some Extensions

Starting with a set of concept names (atomic concepts) and role names (atomic roles), concept descriptions are built using concept constructors. The semantics of concept descriptions is defined using the notion of an interpretation, which assigns sets to concepts and binary relations to roles. First, we introduce the constructors available in the basic description language $\mathcal{ALC}$,[6] together with their semantics.

Definition 1 ($\mathcal{ALC}$ concept descriptions). *Let N_C be a set of* concept names *and N_R a set of* role names. The set of $\mathcal{ALC}$ concept descriptions *is the smallest set such that*

– *all concept names are $\mathcal{ALC}$ concept descriptions;*
– *if C and D are $\mathcal{ALC}$ concept descriptions, then so are $\neg C$, $C \sqcup D$, and $C \sqcap D$;*

[4] http://www.w3.org/TR/2009/WD-owl2-overview-20090327/

[5] http://www.w3.org/TR/owl2-profiles/

[6] Following the usage in the literature, we will sometimes call description languages like $\mathcal{ALC}$ "Description Logics," thereby ignoring the additional ingredients of a DL, such as the terminological formalism.

– *if C is an $\mathcal{ALC}$ concept description and $r \in N_R$, then $\exists r.C$ and $\forall r.C$ are $\mathcal{ALC}$ concept descriptions.*

An interpretation *is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where the* domain *$\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a function that assigns to every concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This function is extended to $\mathcal{ALC}$ concept descriptions as follows:*

– $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
– $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *there is a* $y \in \Delta^{\mathcal{I}}$ *with* $(x, y) \in r^{\mathcal{I}}$ *and* $y \in C^{\mathcal{I}}\}$;
– $(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid$ *for all* $y \in \Delta^{\mathcal{I}}$, $(x, y) \in r^{\mathcal{I}}$ *implies* $y \in C^{\mathcal{I}}\}$.

As usual, the Boolean constructors $\sqcap, \sqcup, \neg$ are respectively called *conjunction*, *disjunction*, and *negation*. We call a concept description of the form $\exists r.C$ an *existential restriction*, and a concept description of the form $\forall r.C$ a *value restriction*. In the following, we will us $\top$ as an abbreviation for $A \sqcup \neg A$, where A is an arbitrary concept name (*top concept*, which is always interpreted as the whole domain), $\bot$ as an abbreviation for $\neg\top$ (*bottom concept*, which is always interpreted as the empty set), and $C \Rightarrow D$ as an abbreviation for $\neg C \sqcup D$ (implication).

The following are examples of $\mathcal{ALC}$ concept descriptions that may be of interest in the conference domain. Assume that $\mathsf{Participant}, \mathsf{Talk}, \mathsf{Boring}, \mathsf{DL}$ are concept names, and $\mathsf{attends}, \mathsf{gives}, \mathsf{topic}$ are role names. The description

$$\mathsf{Participant} \sqcap \exists \mathsf{attends}.\mathsf{Talk}$$

describes conference participants that attend at least on talk,

$$\mathsf{Participant} \sqcap \forall \mathsf{attends}.(\mathsf{Talk} \sqcap \neg\mathsf{Boring})$$

describes conference participants that attend only non-boring talks, and

$$\mathsf{Speaker} \sqcap \exists \mathsf{gives}.(\mathsf{Talk} \sqcap (\mathsf{Boring} \sqcup \forall \mathsf{topic}.\mathsf{DL}))$$

describes speakers giving a talk that is boring or has as its only topic DL.

Relationship with first-order logic. Given the semantics of $\mathcal{ALC}$ concept descriptions, it is easy to see that $\mathcal{ALC}$ can be viewed as a fragment of first-order predicate logic.[7] Indeed, concept names (which are interpreted as sets) are simply unary predicates, and role names (which are interpreted as binary relations) are simply binary predicates. For a given first-order variable x, an $\mathcal{ALC}$ concept description C is translated into a formula $\tau_x(C)$ with free variable x:

– $\tau_x(A) := A(x)$ for concept names A;
– $\tau_x(C \sqcap D) := \tau_x(C) \land \tau_x(D)$;
– $\tau_x(C \sqcup D) := \tau_x(C) \lor \tau_x(D)$;

[7] More information about the connection between DLs and first-order predicate logic can be found in [28].

– $\tau_x(\neg C) := \neg\tau_x(C)$;
– $\tau_x(\forall r.C) := \forall y.(r(x,y) \rightarrow \tau_y(C))$ where y is a variable different from x;
– $\tau_x(\exists r.C) := \exists y.(r(x,y) \wedge \tau_y(C))$ where y is a variable different from x.

Regarding the semantics, any first-order interpretation $\mathcal{I}$ (over the signature consisting of the concept names in N_C as unary predicates and the role names in N_R as binary predicates) can be viewed as an $\mathcal{ALC}$ interpretation and vice versa. Intuitively, the first-order formula $\tau_x(C)$ describes all domain elements $d \in \Delta^{\mathcal{I}}$ that make $\tau_x(C)$ true if x is replaced by them. It is easy to see that this set coincides with the interpretation of the concept description C, i.e.,

$$C^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \mathcal{I} \models \tau_x(C)[x \leftarrow d]\}.$$

The resolution-based approaches for reasoning in DLs are based on such a translation to first-order predicate logic. It should be noted, however, that the translation sketched above does not yield arbitrary first-order formulae. Instead, we obtain formulae belonging to known decidable fragments of first-order predicate logic: the guarded fragment [58] and the two-variable fragment [91,59]. Intuitively, the formulae of the form $\tau_x(C)$ belong to the guarded fragment since every quantified variable y is guarded by a role $r(x,y)$. Regarding membership in the two-variable fragment, it is easy to see that it is enough to use just two first-order variables x, y in the translation: in τ_x one uses y as the variable different from x, and in τ_y one uses x for this purpose.

Relationship with modal logics. There is also a close connection between DLs and modal logics. In particular, $\mathcal{ALC}$ is just a syntactic variant of the basic multimodal logic K [103], where "*multi*modal" means that one has several pairs of box and diamond operators, which are indexed with the name of the corresponding transition relation. In the following, we assume that the reader is familiar with the basic notions of modal logics (see, e.g., [27] for more details). Intuitively, concept names A correspond to propositional variables a and role names r to names for transition relations r. An $\mathcal{ALC}$ concept description C is translated into a modal formula $\theta(C)$ as follows:

– $\theta(A) := a$ for concept names A;
– $\theta(C \sqcap D) := \theta(C) \wedge \theta(D)$;
– $\theta(C \sqcup D) := \theta(C) \vee \theta(D)$;
– $\theta(\neg C) := \neg\theta(C)$;
– $\theta(\forall r.C) := \Box_r\theta(C))$;
– $\theta(\exists r.C) := \Diamond_r\theta(C))$.

Regarding the semantics, any $\mathcal{ALC}$ interpretation $\mathcal{I}$ can be viewed as a Kripke structure $K_{\mathcal{I}}$ (and vice versa): every element w of $\Delta^{\mathcal{I}}$ is a possible world of $K_{\mathcal{I}}$, the world w makes the propositional variable a true iff $w \in A^{\mathcal{I}}$ for the concept name A corresponding to a, and there is a transition from world w to world w' with the transition relation r iff $(w, w') \in r^{\mathcal{I}}$. The translation function θ preserves the semantics in the following sense: $C^{\mathcal{I}}$ is the set of worlds that make $\theta(C)$ true in $K_{\mathcal{I}}$.

The translation-based approaches that reduce reasoning in DLs to reasoning in appropriate modal logics are based on (extensions of) this translation.

Additional constructors. $\mathcal{ALC}$ is only one example of a description language. DL researchers have introduced many additional constructors and investigated various description languages obtained by combining such constructors. Here, we only introduce qualified number restrictions as examples for additional concept constructors, and inverse roles as example for a role constructor (see [5] for an extensive list of additional concept and role constructors).

Qualified number restrictions are of the form $(\geq n\, r.C)$ (at-least restriction) and $(\leq n\, r.C)$ (at-most restriction), where $n \geq 0$ is a non-negative integer, $r \in N_R$ is a role name, and C is a concept description. The semantics of these additional constructors is defined as follows:

$$(\geq n\, r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \mathit{card}(\{e \mid (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \geq n\},$$
$$(\leq n\, r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \mathit{card}(\{e \mid (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}) \leq n\},$$

where $\mathit{card}(X)$ yields the cardinality of the set X. Using qualified number restrictions, we can define the concept of all persons that attend at most 20 talks, of which at least 3 have the topic DL:

$$\mathsf{Person} \sqcap (\leq 20\, \mathsf{attends}.\mathsf{Talk}) \sqcap (\geq 3\, \mathsf{attends}.(\mathsf{Talk} \sqcap \exists \mathsf{topic}.\mathsf{DL})).$$

The *inverse role* constructor applies to a role name r and yields its inverse r^{-1}, where the semantics is the obvious one, i.e.,

$$(r^{-1})^{\mathcal{I}} := \{(e,d) \mid (d,e) \in r^{\mathcal{I}}\}.$$

Inverse roles can be used like role names within concept descriptions. Using the inverse of the role attends, we can define the concept of a speaker giving a boring talk as

$$\mathsf{Speaker} \sqcap \exists \mathsf{gives}.(\mathsf{Talk} \sqcap \forall \mathsf{attends}^{-1}.(\mathsf{Bored} \sqcup \mathsf{Sleeping})).$$

In the following, we will use the notion "concept description" to refer to a description built using the (concept and role) constructors of some description language. Indeed, the definitions of the other three components of a DL (terminological formalism, assertional formalism, reasoning) is independent of the description language. Accordingly, we will also use the notion "role description" to refer to a role name or a role description (such as r^{-1}) built using the constructors of some description language.

2.2 Terminological Knowledge

In its simplest form, a TBox introduces names (abbreviations) for complex descriptions.

Definition 2. *A* concept definition *is of the form $A \equiv C$ where A is a concept name and C is a concept description. Given a set $\mathcal{T}$ of concept definitions, we say that the concept name A* directly uses *the concept name B if $\mathcal{T}$ contains a concept definition $A \equiv C$ such that B occurs in C. Let* uses *be the transitive*

$$\begin{aligned}
\mathsf{Woman} &\equiv \mathsf{Person} \sqcap \mathsf{Female}\\
\mathsf{Man} &\equiv \mathsf{Person} \sqcap \neg\mathsf{Female}\\
\mathsf{Talk} &\equiv \exists\mathsf{topic}.\top\\
\mathsf{Speaker} &\equiv \mathsf{Person} \sqcap \exists\mathsf{gives}.\mathsf{Talk}\\
\mathsf{Participant} &\equiv \mathsf{Person} \sqcap \exists\mathsf{attends}.\mathsf{Talk}\\
\mathsf{BusySpeaker} &\equiv \mathsf{Speaker} \sqcap (\geq 3\,\mathsf{gives}.\mathsf{Talk})\\
\mathsf{BadSpeaker} &\equiv \mathsf{Speaker} \sqcap \forall\mathsf{gives}.(\forall\mathsf{attends}^{-1}.(\mathsf{Bored} \sqcup \mathsf{Sleeping}))
\end{aligned}$$

Fig. 1. A TBox for the conference domain

closure of the relation "directly uses." We say that $\mathcal{T}$ is cyclic *if there is a concept name A that uses itself, and* acyclic *otherwise.*

A TBox *is a finite set $\mathcal{T}$ of concept definitions that is acyclic and such that every concept name occurs at most once on the left-hand side of a concept definition in $\mathcal{T}$. Given a TBox $\mathcal{T}$, we call the concept name A a* defined concept *if A occurs on the left-hand side of a definition in $\mathcal{T}$. All other concept names are called* primitive concepts. *An interpretation $\mathcal{I}$ is a* model *of the TBox $\mathcal{T}$ if it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \equiv C$ in $\mathcal{T}$.*

Fig. 1 shows a small TBox with concept definitions relevant in our example domain. Modern DL systems allow their users to state more general constraints for the interpretation of concepts and roles.

Definition 3. *A* general concept inclusion *axiom (GCI) is of the form $C \sqsubseteq D$ where C, D are concept descriptions. A finite set of GCIs is called a* general TBox*.*

An interpretation $\mathcal{I}$ is a model *of the general TBox $\mathcal{T}$ if it satisfies all its GCIs, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all GCIs $C \sqsubseteq D$ in $\mathcal{T}$.*

Obviously, the concept definition $A \equiv C$ is equivalent (in the sense that is has the same models) to the pair of GCIs $A \sqsubseteq C, C \sqsubseteq A$, which shows that TBoxes can be expressed using general TBoxes. Thus, we assume in the following that the notion of a general TBox subsumes the notion of a TBox. In general, GCIs with complex concept descriptions on their left-hand side cannot be expressed with the help of TBoxes. Using GCIs we can, e.g., say that talks in which all attendants are sleeping are boring

$$\mathsf{Talk} \sqcap \forall\mathsf{attends}^{-1}.\mathsf{Sleeping} \sqsubseteq \mathsf{Boring},$$

and that PC chairs cannot as well be authors

$$\mathsf{Author} \sqcap \mathsf{PCchair} \sqsubseteq \bot.$$

$$\begin{array}{lll} \mathsf{Lecturer(FRANZ)}, & \mathsf{teaches(FRANZ, Tut03)}, & \\ \mathsf{Tutorial(Tut03)}, & \mathsf{topic(Tut03, ReasoningInDL)}, & \mathsf{DL(ReasoningInDL)} \end{array}$$

Fig. 2. An ABox for the conference domain

In some applications, it also makes sense to consider a generalization of TBoxes where one only allows the use of unambiguous definitions, but dispenses with the acyclicity requirement. Such *cyclic TBoxes* $\mathcal{T}$ are thus finite sets of concept definitions such that every concept name occurs at most once on the left-hand side of a concept definition in $\mathcal{T}$ (see, e.g., [4,21] for details).

2.3 Assertional Knowledge

Assertions can be used to state facts about named individuals. Thus, we assume that there is a third set N_I of names, called *individual names*, which is disjoint with the sets of concept and role names. An interpretation additional assigns an element $a^\mathcal{I} \in \Delta^\mathcal{I}$ to every individual name $a \in N_I$.

Definition 4. *Let C be a concept description, r be a role description, and $a, b \in N_I$. An* assertion *is of the form $C(a)$ (concept assertion) or $r(a, b)$ (role assertion). An* ABox *is a finite set of assertions.*

An interpretation $\mathcal{I}$ is a model *of the ABox $\mathcal{A}$ if it satisfies all its assertions, i.e., $a^\mathcal{I} \in C^\mathcal{I}$ holds for all concept assertions $C(a) \in \mathcal{A}$ and $(a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I}$ holds for all role assertions $r(a, b) \in \mathcal{A}$.*

Fig. 2 shows a small ABox with assertions describing a specific DL tutorial.

2.4 Inference Problems

DL systems provide their users with inference capabilities that allow them to derive implicit knowledge from the one explicitly represented. The following are the most important "classical" inference problems supported by DL systems.

Definition 5. *Let $\mathcal{T}$ be a generalized TBox, $\mathcal{A}$ an ABox, C, D concept descriptions, and a an individual name.*

- *C is* subsumed by *D* w.r.t. *$\mathcal{T}$ ($C \sqsubseteq_\mathcal{T} D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*
- *C is* equivalent to *D* w.r.t. *$\mathcal{T}$ ($C \equiv_\mathcal{T} D$) iff $C^\mathcal{I} = D^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$.*
- *C is* satisfiable w.r.t. *$\mathcal{T}$ iff $C^\mathcal{I} \neq \emptyset$ for some model $\mathcal{I}$ of $\mathcal{T}$.*
- *$\mathcal{A}$ is* consistent w.r.t. *$\mathcal{T}$ iff it has a model that is also a model of $\mathcal{T}$.*
- *a is an* instance of *C* w.r.t. *$\mathcal{A}$* and *$\mathcal{T}$ ($\mathcal{A} \models_\mathcal{T} C(a)$) iff $a^\mathcal{I} \in C^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$.*

One might think that, in order to realize the inference component of a DL system, on needs to design and implement five algorithms, each solving one of the above inference problems. Fortunately, this is not the case since there exist the following polynomial time reductions, which only require the availability of the concept constructors conjunction and negation in the description language:

– Subsumption can be be reduced in polynomial time to equivalence:

$$C \sqsubseteq_{\mathcal{T}} D \text{ iff } C \sqcap D \equiv_{\mathcal{T}} C$$

– Equivalence can be be reduced in polynomial time to subsumption:

$$C \equiv_{\mathcal{T}} D \text{ iff } C \sqsubseteq_{\mathcal{T}} D \text{ and } D \sqsubseteq_{\mathcal{T}} C$$

– Subsumption can be be reduced in polynomial time to (un)satisfiability:

$$C \sqsubseteq_{\mathcal{T}} D \text{ iff } C \sqcap \neg D \text{ is unsatisfiable w.r.t. } \mathcal{T}$$

– Satisfiability can be be reduced in polynomial time to (non-)subsumption:

$$C \text{ is satisfiable w.r.t. } \mathcal{T} \text{ iff not } C \sqsubseteq_{\mathcal{T}} \bot$$

– Satisfiability can be be reduced in polynomial time to consistency:

$$C \text{ is satisfiable w.r.t. } \mathcal{T} \text{ iff } \{C(a)\} \text{ is consistent w.r.t. } \mathcal{T}$$

– The instance problem can be reduced in polynomial time to (in)consistency:

$$\mathcal{A} \models_{\mathcal{T}} C(a) \text{ iff } \mathcal{A} \cup \{\neg C(a)\} \text{ is inconsistent w.r.t. } \mathcal{T}$$

– Consistency can be reduced in polynomial time to the (non-)instance problem:

$$\mathcal{A} \text{ is consistent w.r.t. } \mathcal{T} \text{ iff } \mathcal{A} \not\models_{\mathcal{T}} \bot(a)$$

Thus, if one is only interested in terminological reasoning (i.e., satisfiability, equivalence, and subsumption), it is enough to have a satisfiability algorithm. If one is additionally interested in assertional reasoning (i.e., consistency and instance), then it is enough to have a consistency algorithm.

Another important observation is that reasoning w.r.t. a normal (i.e., not general) TBox can be reduced to reasoning w.r.t. the empty TBox.[8] Intuitively, TBoxes merely state that defined concepts are abbreviations for certain complex concept descriptions. These complex descriptions can be made explicit by *expanding* the definitions from $\mathcal{T}$: given a concept description C, its expansion $exp(C, \mathcal{T})$ w.r.t. $\mathcal{T}$ is obtained by exhaustively replacing all defined concept names A occurring on the left-hand side of concept definitions $A \equiv C$ by their defining concept descriptions C. For example, w.r.t. the TBox of Fig. 1, the concept description $\mathsf{Woman} \sqcap \mathsf{BusySpeaker}$ is expanded to

$$\mathsf{Person} \sqcap \mathsf{Female} \sqcap \mathsf{Person} \sqcap \exists \mathsf{gives.Talk} \sqcap (\geq 3\, \mathsf{gives.Talk}),$$

which is equivalent to $\mathsf{Person} \sqcap \mathsf{Female} \sqcap (\geq 3\, \mathsf{gives.Talk})$.

[8] Instead of saying "w.r.t. $\emptyset$" one usually says "without a TBox," and omits the index $\mathcal{T}$ for subsumption, equivalence, and instance, i.e., writes $\equiv$, $\sqsubseteq$, $\models$ instead of $\equiv_{\mathcal{T}}$, $\sqsubseteq_{\mathcal{T}}$, and $\models_{\mathcal{T}}$.

$$
\begin{aligned}
A_0 &\equiv \forall r.A_1 \sqcap \forall s.A_1 \\
A_1 &\equiv \forall r.A_2 \sqcap \forall s.A_2 \\
&\vdots \\
A_{n-1} &\equiv \forall r.A_n \sqcap \forall s.A_n
\end{aligned}
$$

Fig. 3. A TBox $\mathcal{T}$ that causes exponential blow-up during expansion

It is easy to show that $C \sqsubseteq_{\mathcal{T}} D$ iff $exp(C, \mathcal{T}) \sqsubseteq exp(D, \mathcal{T})$. Similar reduction are possible for the other inference problems. It should be noted, however, that these reductions are in general exponential. For example, expanding the concept description A_0 w.r.t. the TBox of Fig. 3 yields an expanded description $exp(A_0, \mathcal{T})$ that contains the concept name A_n 2^n times. This exponential blow-up can sometimes be avoided by devising satisfiability algorithms that explicitly take acyclic TBoxes into account. For example, satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. TBoxes is PSpace-complete, and without TBoxes this problem is of exactly the same complexity [107,83]. However this is not always the case: in Section 4, we will introduce the DL $\mathcal{FL}_0$, for which reasoning w.r.t. TBoxes is considerably more difficult than reasoning without them [94].

For some expressive DLs it is possible to reduce reasoning w.r.t. a general TBox to reasoning without a TBox [10,70], but for $\mathcal{ALC}$ this is not possible, i.e., one really needs to design algorithms that take GCIs into account.

Compound inferences. Some of the most important inference problems in DLs are of a compound nature in the sense that, in principle, they can be reduced to multiple invocations of the more basic inference problems mentioned above. However, when the goal is to achieve an efficient implementation, it is vital to consider compound inferences as first-class citizens since a naïve reduction to the basic inference problems may be too inefficient [12]. Here, we define two of these compound inference problems, but do not deal with the efficiency issue.

Classification. Given a (general) TBox $\mathcal{T}$, compute the restriction of the subsumption relation "$\sqsubseteq_{\mathcal{T}}$" to the set of concept names used in $\mathcal{T}$.

Realization. Given an ABox $\mathcal{A}$, a (general) TBox $\mathcal{T}$, and an individual name a, compute the set $R_{\mathcal{A},\mathcal{T}}(a)$ of those concept names A that are used in $\mathcal{T}$, satisfy $\mathcal{A} \models_{\mathcal{T}} A(a)$, and are minimal with this property w.r.t. the subsumption relation "$\sqsubseteq_{\mathcal{T}}$".

Complexity of reasoning. In the 1980ies, it was a commonly held belief that reasoning in knowledge representation systems should be tractable, i.e., of polynomial time complexity. The precursor of all DL systems, Klone [33], as well as its early successor systems, like K-Rep [88], Back [100], and Loom [87], indeed employed polynomial-time subsumption algorithms. Later on, however, it turned out that subsumption in rather inexpressive DLs may be intractable [82], that subsumption in Klone is even undecidable [106], and that subsumption w.r.t. a TBox in a description language with conjunction ($\sqcap$) and value restriction ($\forall r.C$)[9] is intractable [94]. The reason for the discrepancy between the complexity of the subsumption algorithms employed in the above mention early DL systems and the worst-case complexity of the subsumption problems these algorithms were supposed to solve was, as mentioned in the introduction, due to the fact that these systems employed sound, but incomplete subsumption algorithms, i.e., algorithms whose positive answers to subsumption queries are correct, but whose negative answers may be incorrect.

The use of incomplete algorithms has since then largely been abandoned in the DL community, mainly because of the problem that the behavior of the systems is no longer determined by the semantics of the description language: an incomplete algorithm may claim that a subsumption relationship does not hold, although it should hold according to the semantics. This left the DL community with two ways out of the complexity dilemma:

– Employ expressive DLs with sound and complete, but intractable inference procedures.
– Employ inexpressive DLs that allow the use of sound, complete, and tractable inference procedures.

In the next two sections, we treat these two approaches in more detail.

It should be noted that here we have barely scratched the surface of the research on the complexity of reasoning in DLs. Indeed, DL researchers have investigated the complexity of reasoning in a great variety of DLs in detail. Giving an overview of the results obtained in this direction in the last 20 years is beyond the scope of this article. We refer the reader to overview articles such as [47,18] and the Description Logic Complexity Navigator[10] for more details.

3 Reasoning in Expressive DLs

As mentioned in the introduction, a variety of of reasoning techniques have been introduced for expressive DLs. Here, we describe tableau-based and automata-based approaches in some detail, but do not treat approaches based on translations to first-order or modal logic.

Before looking at specific inference procedures in detail, let us first state some general requirements on the behavior of such procedures:

– The procedure should be a *decision procedure* for the problem, which means that it should be:
  • *sound*, i.e., the positive answers should be correct;
  • *complete*, i.e., the negative answers should be correct;
  • *terminating*, i.e., it should always give an answer in finite time
– The procedure should be as *efficient* as possible. Preferably, it should be optimal w.r.t. the (worst-case) complexity of the problem.

[9] All the systems mentioned above supported these two concept constructors, which were at that time viewed as being indispensable for a DL.

[10] http://www.cs.man.ac.uk/~ezolin/dl/

– The procedure should be *practical*, i.e., easy to implement and optimize, and behave well in applications.

Both tableau-based and automata-based approaches to reasoning in DLs yield decision procedures. Tableau-based approaches often yield practical procedures: optimized implementations of such procedures have turned out to behave quite well in applications even for expressive DLs with a high worst-case complexity. However, these practical procedures are often not optimal w.r.t. the worst-case complexity of the problem: in particular, satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes is ExpTime-complete, but it is very hard to design a tableau-based procedure for it that runs in deterministic exponential time. In contrast, it is quite easy to design an ExpTime automata-based procedure for this problem, but there are no practical implementations for this procedure.

3.1 Tableau-Based Approaches

The most widely used reasoning technique for DLs is the tableau-based approach, which was first introduced in the context of DLs by Schmidt-Schauß and Smolka [108], though it had already been used for modal logics long before that [53]. In this section, we first describe this technique for the case of consistency of an ABox (without a TBox[11]) in our basic DL $\mathcal{ALC}$. Then we show how the approach can be extended to deal with qualified number restrictions and with general TBoxes.

Given an $\mathcal{ALC}$ ABox $\mathcal{A}_0$, the tableau algorithm for consistency tries to construct a finite interpretation $\mathcal{I}$ that is a model of $\mathcal{A}_0$. Before we can describe the algorithm more formally, we need to introduce an appropriate data structure in which to represent the (partial descriptions of) finite interpretations that are generated during the run of the algorithm. The original paper by Schmidt-Schauß and Smolka [108], and also many other papers on tableau algorithms for DLs, introduce the new notion of a constraint system for this purpose. However, if we look at the information that must be expressed (namely, the elements of the interpretation, the concept descriptions they belong to, and their role relationships), we see that ABox assertions are sufficient for this purpose.

It will be convenient to assume that all concept descriptions are in *negation normal form* (NNF), i.e., that negation occurs only directly in front of concept names. Using de Morgan's rules and the usual rules for quantifiers, any $\mathcal{ALC}$ concept description can be transformed (in linear time) into an equivalent description in NNF. An ABox is in NNF if all the concept descriptions occurring in it are in NNF.

Let $\mathcal{A}_0$ be an $\mathcal{ALC}$ ABox in NNF. In order to test consistency of $\mathcal{A}_0$, the algorithm starts with $\mathcal{A}_0$, and applies consistency preserving transformation rules (see Fig. 4) to this ABox. The transformation rule that handles disjunction is *nondeterministic* in the sense that a given ABox is transformed into two new ABoxes such that the original ABox is consistent iff *one of* the new ABoxes is so.

[11] As mentioned above, inference problems w.r.t. a TBox can be reduced to the corresponding ones without TBoxes by expanding the concept definitions from $\mathcal{T}$.

The $\rightarrow_\sqcap$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

The $\rightarrow_\sqcup$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{C_2(x)\}$.

The $\rightarrow_\exists$-rule
Condition: $\mathcal{A}$ contains $(\exists r.C)(x)$, but there is no individual name z such that $C(z)$ and $r(x,z)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y), r(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

The $\rightarrow_\forall$-rule
Condition: $\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x,y)$, but it does not contain $C(y)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$.

Fig. 4. Tableau rules of the consistency algorithm for $\mathcal{ALC}$

For this reason we will consider finite sets of ABoxes $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ instead of single ABoxes. Such a set is *consistent* iff there is some i, $1 \leq i \leq k$, such that $\mathcal{A}_i$ is consistent. A rule of Fig. 4 is applied to a given finite set of ABoxes $\mathcal{S}$ as follows: it takes an element $\mathcal{A}$ of $\mathcal{S}$, and replaces it by one ABox $\mathcal{A}'$ or by two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$.

Definition 6. *An ABox $\mathcal{A}$ is called* complete *iff none of the transformation rules of Fig. 4 applies to it. The ABox $\mathcal{A}$ contains a* clash *iff $\{A(x), \neg A(x)\} \subseteq \mathcal{A}$ for some individual name x and some concept name A. An ABox is called* closed *if it contains a clash, and* open *otherwise.*

The *consistency algorithm for $\mathcal{ALC}$* works as follows. It starts with the singleton set of ABoxes $\{\mathcal{A}_0\}$, and applies the rules of Fig. 4 (in arbitrary order) until no more rules apply. It answers "consistent" if the set $\widehat{\mathcal{S}}$ of ABoxes obtained this way contains an open ABox, and "inconsistent" otherwise. The fact that this algorithm is a decision procedure for consistency of $\mathcal{ALC}$ ABoxes is an easy consequence of the following lemma.

Lemma 1. *Let $\mathcal{A}_0$ be an $\mathcal{ALC}$ ABox in negation normal form.*

1. Local correctness: *the rules preserve consistency, i.e., if $\mathcal{S}'$ is obtained from the finite set of ABoxes $\mathcal{S}$ by application of a transformation rule, then $\mathcal{S}$ is consistent iff $\mathcal{S}'$ is consistent.*
2. Termination: *there cannot be an infinite sequence of rule applications*

$$\{\mathcal{A}_0\} \rightarrow \mathcal{S}_1 \rightarrow \mathcal{S}_2 \rightarrow \cdots .$$

3. Soundness:[12] *any complete and open ABox $\mathcal{A}$ is consistent.*
4. Completeness:[13] *any closed ABox $\mathcal{A}$ is inconsistent.*

Proof 1. Local correctness: We treat the $\rightarrow_\exists$-rule and the $\rightarrow_\sqcup$-rule in detail. The other rules can be handled similarly.

First, assume that $\mathcal{S}'$ is obtained from $\mathcal{S}$ by an application of the $\rightarrow_\exists$-rule. Then there is an ABox $\mathcal{A} \in \mathcal{S}$ containing an assertion of the form $(\exists r.C)(x)$, and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing $\mathcal{A}$ by $\mathcal{A}' := \mathcal{A} \cup \{C(y), r(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

Obviously, it is enough to show that $\mathcal{A}$ has a model iff $\mathcal{A}'$ has a model. The if-direction is trivial since $\mathcal{A} \subseteq \mathcal{A}'$. To show the only-if direction, assume that $\mathcal{I}$ is a model of $\mathcal{A}$. Since $(\exists r.C)(x) \in \mathcal{A}$, there is a $d \in \Delta^\mathcal{I}$ such that

$$(x^\mathcal{I}, d) \in r^\mathcal{I} \;\text{ and }\; d \in C^\mathcal{I}.$$

Let $\mathcal{I}'$ be the interpretation that coincides with $\mathcal{I}$, with the exception that $y^{\mathcal{I}'} = d$. Since y does not occur in $\mathcal{A}$, $\mathcal{I}'$ is a model of $\mathcal{A}$. By the definition of $y^{\mathcal{I}'}$, it is also a model of $\{r(x,y), C(y)\}$, and thus of $\mathcal{A}'$.

Second, assume that $\mathcal{S}'$ is obtained from $\mathcal{S}$ by an application of the $\rightarrow_\sqcup$-rule. Then there is an ABox $\mathcal{A} \in \mathcal{S}$ containing an assertion of the form $(C_1 \sqcup C_2)(x)$, and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing $\mathcal{A}$ by $\mathcal{A}' := \mathcal{A} \cup \{C_1(x)\}$ and $\mathcal{A}'' := \mathcal{A} \cup \{C_2(x)\}$.

It is enough to show that $\mathcal{A}$ has a model iff $\mathcal{A}'$ has a model or $\mathcal{A}''$ has a model. The if-direction is again trivial since $\mathcal{A} \subseteq \mathcal{A}'$ and $\mathcal{A} \subseteq \mathcal{A}''$. To show the only-if direction, assume that $\mathcal{I}$ is a model of $\mathcal{A}$. Since $(C_1 \sqcup C_2)(x) \in \mathcal{A}$, we have

$$x^\mathcal{I} \in (C_1 \sqcup C_2)^\mathcal{I} = C_1^\mathcal{I} \cup C_2^\mathcal{I}.$$

If $x^\mathcal{I} \in C_1^\mathcal{I}$, then $\mathcal{I}$ is a model of $\mathcal{A}'$. If $x^\mathcal{I} \in C_2^\mathcal{I}$, then $\mathcal{I}$ is a model of $\mathcal{A}''$.

2. Termination: Define the label $\mathcal{L}_\mathcal{A}(x)$ of an individual name x in an ABox $\mathcal{A}$ to consist of the concept descriptions in concept assertions for x, i.e.,

$$\mathcal{L}_\mathcal{A}(x) := \{C \mid C(x) \in \mathcal{A}\}.$$

Let $\mathcal{S}$ be a set of ABoxes reached by a finite number of rule applications, starting with $\{\mathcal{A}_0\}$, and let $\mathcal{A} \in \mathcal{S}$. The following are easy consequences of the definition of the tableau rules.

1. rule application is monotonic, i.e., every application of a rule to $\mathcal{A}$ extends the label of an individual, by adding a new concept assertion, and does not remove any element from a label;

[12] Recall that *soundness* means that the positive answers of the algorithm are correct, i.e., if the algorithm says "consistent," then the input ABox $\mathcal{A}_0$ is indeed consistent. This follows from the part 3. of the lemma together with part 1. (local correctness).

[13] Recall that *completeness* means that the negative answers of the algorithm are correct, i.e., if the algorithm says "inconsistent," then the input ABox $\mathcal{A}_0$ is indeed inconsistent. This follows from the part 4. of the lemma together with part 1. (local correctness).

2. concept descriptions occurring in labels in $\mathcal{A}$ are subdescriptions of concept descriptions occurring in the initial ABox $\mathcal{A}_0$.

Clearly, these two facts imply that there can only be a finite number of rule applications per individual. Thus, it remains to show that the number of newly introduced individuals in a chain of rule applications is bounded as well. Let us call an individual name occurring in $\mathcal{A}$ a *new* individual if it is not one of the individuals already present in $\mathcal{A}_0$. We say that y is an r-successor of x if $r(x, y) \in \mathcal{A}$.

3. for a given individual x, an existential restriction in the label of x can trigger at most one introduction of a new individual, and thus the number of new individuals that are r-successors of an individual in $\mathcal{A}$ is bounded by the number of existential restrictions in $\mathcal{A}_0$;
4. the length of successor chains of new individuals in $\mathcal{A}$ is bounded by the maximal size of the concept descriptions occurring in $\mathcal{A}_0$. This is an immediate consequence of the following two facts:
   - if x is a new individual in $\mathcal{A}$, then it has a unique predecessor y
   - the maximal size of concept descriptions in $\mathcal{L}_{\mathcal{A}}(x)$ is strictly smaller than the maximal size of concept descriptions in $\mathcal{L}_{\mathcal{A}}(y)$.

Facts 3. and 4. yield an overall bound on the number of new individuals in $\mathcal{A}$. Since only a finite number of individuals can be introduced during rule application, and only finitely many rules can be applied to a fixed individual, this shows that overall we can have only a finite number of rule applications, which completes the proof of termination.

3. Soundness: Let $\mathcal{A}$ be a complete and open ABox. To prove that $\mathcal{A}$ is consistent, we define the *canonical interpretation* $\mathcal{I}_{\mathcal{A}}$, and show that it is a model of $\mathcal{A}$:

1. The domain $\Delta^{\mathcal{I}_{\mathcal{A}}}$ of $\mathcal{I}_{\mathcal{A}}$ consists of the individual names occurring in $\mathcal{A}$.
2. For all individual names x we define $x^{\mathcal{I}_{\mathcal{A}}} := x$.
3. For all concept names A we define $A^{\mathcal{I}_{\mathcal{A}}} := \{x \mid A(x) \in \mathcal{A}\}$.
4. For all role names r we define $r^{\mathcal{I}_{\mathcal{A}}} := \{(x, y) \mid r(x, y) \in \mathcal{A}\}$.

By definition, $\mathcal{I}_{\mathcal{A}}$ satisfies all the role assertions in $\mathcal{A}$. To prove that $\mathcal{I}_{\mathcal{A}}$ satisfies the concept assertions as well, we consider $C(x) \in \mathcal{A}$ and show $x^{\mathcal{I}_{\mathcal{A}}} = x \in C^{\mathcal{I}_{\mathcal{A}}}$ by *induction* on the size of C:

- $C = A$ for $A \in N_C$: $x \in A^{\mathcal{I}_{\mathcal{A}}}$ is an immediate consequence of the definition of $A^{\mathcal{I}_{\mathcal{A}}}$.
- $C = \neg A$ for $A \in N_C$: since $\mathcal{A}$ is *open*, $A(x) \notin \mathcal{A}$, and thus $x \notin A^{\mathcal{I}_{\mathcal{A}}}$ by the definition of $A^{\mathcal{I}_{\mathcal{A}}}$.
- $C = C_1 \sqcap C_2$: since $\mathcal{A}$ is *complete*, $(C_1 \sqcap C_2)(x) \in \mathcal{A}$ implies that $C_1(x) \in \mathcal{A}$ and $C_2(x) \in \mathcal{A}$; by *induction*, this yields $x \in C_1^{\mathcal{I}_{\mathcal{A}}}$ and $x \in C_2^{\mathcal{I}_{\mathcal{A}}}$, and thus $x \in (C_1 \sqcap C_2)^{\mathcal{I}_{\mathcal{A}}}$.
- the other constructors can be treated similarly.

4. Completeness: the fact that a closed ABox cannot have a model is an immediate consequence of the definition of a clash. □

Theorem 1. *The tableau algorithm introduced above is a decision procedure for consistency of $\mathcal{ALC}$ ABoxes.*

Proof. Started with a finite $\mathcal{ALC}$ ABox $\mathcal{A}_0$ in NNF, the algorithm always terminates with a finite set of complete ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Local correctness implies that $\mathcal{A}_0$ is consistent iff one of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is consistent.

If the algorithm answers "inconsistent," then all the ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are closed. Completeness then yields that all the ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are inconsistent, and thus $\mathcal{A}_0$ is inconsistent, by local correctness.

If the algorithm answers "consistent," then one of the complete ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$, say $\mathcal{A}_i$, is open. Soundness then yields that $\mathcal{A}_i$ is consistent, and thus $\mathcal{A}_0$ is consistent, by local correctness.

To sum up, we have shown that the algorithm always terminates, and that both the positive answers ("consistent") and the negative answers ("inconsistent") are correct. □

Adding qualified number restrictions. The description language obtained from $\mathcal{ALC}$ by adding qualified number restrictions is called $\mathcal{ALCQ}$. In order to transform also $\mathcal{ALCQ}$ ABoxes into negation normal form, we additionally use the following equivalence preserving rules:

$$\begin{aligned} \neg(\geq n+1\, r.C) &\rightsquigarrow (\leq n\, r.C) \\ \neg(\geq 0\, r.C) &\rightsquigarrow \bot \\ \neg(\leq n\, r.C) &\rightsquigarrow (\geq n+1\, r.C) \end{aligned}$$

In the following, we assume that all $\mathcal{ALCQ}$ ABoxes are in NNF.

The main idea underlying the extension of the tableau algorithm for $\mathcal{ALC}$ to $\mathcal{ALCQ}$ is quite simple. At-least restrictions are treated by generating the required role successors as new individuals. At-most restrictions that are currently violated are treated by (non-deterministically) identifying some of the role successors. To avoid running into a generate-identify cycle, we introduce explicit inequality assertions that prohibit the identification of individuals that were introduced to satisfy the same at-least restriction. This use of inequality assertions also creates new types of clashes, which occur when an at-most restriction requires some identification, but all identifications are prohibited by inequality assertions.

To be more precise, the tableau algorithm for consistency of $\mathcal{ALC}$ ABoxes is extended to $\mathcal{ALCQ}$ as follows:

– For each of the new concept constructors, we add a *new tableau rule:* the $\rightarrow_{\geq}$-rule and the $\rightarrow_{\leq}$-rule are shown in Fig. 5.
– In the formulation of these rules, we have used *inequality assertions*, which are of the form $x \neq y$ for individual names x, y, and have the obvious semantics that an interpretation $\mathcal{I}$ satisfies such an assertion iff $x^{\mathcal{I}} \neq y^{\mathcal{I}}$.
– Finally, there are *new types of clashes*:

The $\rightarrow_{\geq}$-rule

Condition: $\mathcal{A}$ contains $(\geq n\, r.C)(x)$, and there are no individual names $z_1, \ldots, z_n$ such that $r(x, z_i), C(z_i)$ $(1 \leq i \leq n)$ and $z_i \not\doteq z_j$ $(1 \leq i < j \leq n)$ are in $\mathcal{A}$.

Action: $\mathcal{A}' := \mathcal{A} \cup \{r(x, y_i), C(y_i) \mid 1 \leq i \leq n\} \cup \{y_i \not\doteq y_j \mid 1 \leq i < j \leq n\}$, where $y_1, \ldots, y_n$ are distinct individual names not occurring in $\mathcal{A}$.

The $\rightarrow_{\leq}$-rule

Condition: $\mathcal{A}$ contains distinct individual names $y_1, \ldots, y_{n+1}$ such that $(\leq n\, r.C)(x)$ and $r(x, y_1), C(y_1) \ldots, r(x, y_{n+1}), C(y_{n+1})$ are in $\mathcal{A}$, and $y_i \not\doteq y_j$ is not in $\mathcal{A}$ for some $i, j, 1 \leq i < j \leq n + 1$.

Action: For each pair y_i, y_j such that $1 \leq i < j \leq n + 1$ and $y_i \not\doteq y_j$ is not in $\mathcal{A}$, the ABox $\mathcal{A}_{i,j} := [y_i/y_j]\mathcal{A}$ is obtained from $\mathcal{A}$ by replacing each occurrence of y_i by y_j.

Fig. 5. Tableau rules for qualified number restrictions

- $x \not\doteq x \in \mathcal{A}$ for an individual name x.
- $\{(\leq n\, r.C)(x)\} \cup \{r(x, y_i), C(y_i) \mid 1 \leq i \leq n + 1\} \cup \{y_i \not\doteq y_j \mid 1 \leq i < j \leq n + 1\} \subseteq \mathcal{A}$ for individual names $x, y_1, \ldots, y_{n+1}$, an $\mathcal{ALCQ}$ concept description C, a role name r, and a non-negative integer n.

The main question is then, of course, whether this extended algorithm really yields a decision procedure for consistency of $\mathcal{ALCQ}$ ABoxes. To prove this, it would be enough to show that the four properties stated in Lemma 1 also hold for the extended algorithm. Local correctness and completeness are easy to show. Unfortunately, neither soundness nor termination hold.

To see that the algorithm is not sound, consider the ABox

$$\mathcal{A}_0 := \{(\geq 3\, \mathsf{child}.\top)(x),\ (\leq 1\, \mathsf{child}.\mathsf{Female})(x),\ (\leq 1\, \mathsf{child}.\neg\mathsf{Female})(x)\}.$$

Obviously, this ABox is inconsistent, but the algorithm does not find this out. In fact, it would introduce three new individuals y_1, y_2, y_3 as r-successors of x, each belonging to $\top$. In an interpretation, the element $y_i^{\mathcal{I}}$ $(i = 1, 2, 3)$ belongs to either $\mathsf{Female}^{\mathcal{I}}$ or to $(\neg\mathsf{Female})^{\mathcal{I}}$, but in the ABox $\mathcal{A}_1$ obtained by applying the $\rightarrow_{\geq}$-rule to $\mathcal{A}_0$, the only concept assertion for y_i $(i = 1, 2, 3)$ is $\top(y_i)$. Thus, the $\rightarrow_{\leq}$-rule does not apply to $\mathcal{A}_1$, and $\mathcal{A}_1$ also does not contain a clash. The ABox $\mathcal{A}_1$ is thus complete and open, but it is not consistent.

The soundness problem illustrated by this example can be avoided by adding as a third rule the $\rightarrow_{\text{choose}}$-rule shown in Fig. 6, where $\sim C$ denotes the negation normal form of C. It is easy to show that this rule preserves local correctness, and that its addition allows us to regain soundness.

However, we still need to deal with the termination problem. This problem is illustrated in the following example. Consider the ABox

$$\mathcal{A}_0 := \{A(a),\ r(a, a),\ (\exists r.A)(a),\ (\leq 1\, r.\top)(a),\ (\forall r.\exists r.A)(a), r(a, x),\ A(x)\}.$$

The $\rightarrow_{\text{choose}}$-rule
Condition: $\mathcal{A}$ contains $(\leq n\, r.C)(x)$ and $r(x,y)$, but neither $C(y)$ nor $\neg C(y)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{{\sim}C(y)\}$.

Fig. 6. The $\rightarrow_{\text{choose}}$-rule for qualified number restrictions

The $\rightarrow_{\forall}$-rule can be used to add the assertion $(\exists r.A)(x)$, which yields the new ABox

$$\mathcal{A}_1 := \mathcal{A}_0 \cup \{(\exists r.A)(x)\}.$$

This triggers an application of the $\rightarrow_{\exists}$-rule to x. Thus, we obtain the new ABox

$$\mathcal{A}_2 := \mathcal{A}_1 \cup \{r(x,y),\ A(y)\}.$$

Since a has two r-successors in $\mathcal{A}_2$, the $\rightarrow_{\leq}$-rule is applicable to a. By replacing every occurrence of x by a, we obtain the ABox

$$\mathcal{A}_3 := \{A(a),\ r(a,a),\ (\exists r.A)(a),\ (\leq 1\, r.\top)(a),\ (\forall r.\exists r.A)(a), r(a,y),\ A(y)\},$$

Except for the individual names (i.e., y instead of x), $\mathcal{A}_3$ is identical to $\mathcal{A}_1$. For this reason, we can continue as above to obtain an infinite chain of rule applications.

We can easily regain termination by requiring that *generating rules* (i.e., the rules $\rightarrow_{\exists}$ and $\rightarrow_{\geq}$, which generate new individuals) may only be applied if none of the other rules is applicable. In the above example, this strategy would prevent the application of the $\rightarrow_{\exists}$-rule to x in the ABox $\mathcal{A}_1$ since the $\rightarrow_{\leq}$-rule is also applicable. After applying the $\rightarrow_{\leq}$-rule (which replaces x by a), the $\rightarrow_{\exists}$-rule is no longer applicable since a already has an r-successor that belongs to A.

Consistency w.r.t. general TBoxes. Let $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ be a general TBox. It is easy to see that the general TBox consisting of the single GCI

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap \ldots \sqcap (\neg C_n \sqcup D_n)$$

is equivalent to $\mathcal{T}$ in the sense that it has the same models. Thus, it is sufficient to deal with the case where the general TBox consists of a single GCI of the form $\top \sqsubseteq C$ for a concept description C. Obviously, this GCI says that every element of the model belongs to C. Thus, to reason w.r.t. a general TBox consisting of this GCI, it makes sense to add a new rule, the $\rightarrow_{\top\sqsubseteq C}$-rule, which adds the concept assertion $C(x)$ in case the individual name x occurs in the ABox, and this assertion is not yet present.

Does the addition of the $\rightarrow_{\top\sqsubseteq C}$-rule yield a decision procedure for ABox consistency w.r.t. the general TBox $\{\top \sqsubseteq C\}$? Local correctness, soundness, and completeness can indeed easily be shown, but the procedure does not terminate, as illustrated by the following example. Consider the ABox $\mathcal{A}_0 := \{(\exists r.A)(x_0)\}$, and assume that we want to test its consistency w.r.t. the general TBox $\{\top \sqsubseteq \exists r.A\}$. The procedure generates an infinite sequence of ABoxes $\mathcal{A}_1, \mathcal{A}_2, \ldots$ and individuals $x_1, x_2, \ldots$ such that $\mathcal{A}_{i+1} := \mathcal{A}_i \cup \{r(x_i, x_{i+1}),\ A(x_{i+1}),\ (\exists r.A)(x_{i+1})\}$. Since all individuals x_i $(i \geq 1)$ receive the same concept assertions as x_1, we may say that the procedure has run into a cycle.

Termination can be regained by using a mechanism that detects cyclic computations, and then blocking the application of generating rules: the application of the $\rightarrow_\exists$- and the $\rightarrow_\geq$-rule to an individual x is *blocked* by an individual y in an ABox $\mathcal{A}$ iff $\mathcal{L}_\mathcal{A}(x) \subseteq \mathcal{L}_\mathcal{A}(y)$.[14] The main idea underlying blocking is that the blocked individual x can use the role successors of y instead of generating new ones. For example, instead of generating a new r-successor for x_2 in the above example, one can simply use the r-successor of x_1. This yields an interpretation $\mathcal{I}$ with $\Delta^\mathcal{I} := \{x_0, x_1, x_2\}$, $A^\mathcal{I} := \{x_1, x_2\}$, and $r^\mathcal{I} := \{(x_0, x_1),\ (x_1, x_2), (x_2, x_2)\}$. Obviously, $\mathcal{I}$ is a model of both $\mathcal{A}_0$ and the general TBox $\{\top \sqsubseteq \exists r.A\}$.

To avoid cyclic blocking (of x by y and vice versa), we consider an enumeration of all individual names, and require that an individual x may only be blocked by individuals y that occur before x in this enumeration. This, together with some other technical assumptions, makes sure that a tableau algorithm using this notion of blocking is sound and complete as well as terminating both for $\mathcal{ALC}$ and $\mathcal{ALCQ}$ (see, e.g., [37,9] for details).

Complexity of reasoning. For $\mathcal{ALC}$, the satisfiability and the consistency problem (without TBox) are PSpace-complete [107,64]. The tableau algorithm as described above needs exponential space, but it can be modified such that it needs only polynomial space [107]. Both TBoxes [83] and qualified number restrictions [65,113] can be added without increasing the complexity. W.r.t. general TBoxes, the satisfiability and the consistency problem are ExpTime-complete [103]. However, it is not easy to show the ExpTime-upper bound using tableau algorithms, though it is in principle possible [48,56]. As we will see in the next section, automata-based algorithms are well-suited to show such ExpTime-upper bounds. The tableau algorithms implemented in systems like FaCT, Racer, and Pellet are not worst-case optimal, but they are nevertheless highly optimized and behave quite well on large knowledge bases from applications.

3.2 Automata-Based Approaches

Although the tableau-based approach is currently the most widely used technique for reasoning in DLs, other approaches have been developed as well. In general, a reasoning algorithm may be developed with different intentions in mind, such as using it for an optimized implementation or using it to prove a decidability or computational complexity result. Certain approaches may (for a given logic) be better suited for the former task, whereas others may be better suited for the latter—and it is sometimes hard to find one that is well-suited for both. As mentioned above, the tableau-based approach often yields practical algorithms, whereas it is not well-suited for proving ExpTime-upper bounds. In

[14] Recall that $\mathcal{L}_\mathcal{A}(z) = \{C \mid C(z) \in \mathcal{A}\}$ for any individual z occurring in $\mathcal{A}$.

contrast, such upper bounds can often be shown in a very elegant way using automata-based approach [41,86,84,114].[15]

In this subsection, we restrict our attention to concept satisfiability, possibly w.r.t. (general) TBoxes. This is not a severe restriction since most of the other interesting inference problems can be reduced to satisfiability.[16] There are various instances of the automata-based approach, which differ not only w.r.t. the DL under consideration, but also w.r.t. the employed automaton model. However, in principle all these instances have the following general ideas in common:

- First, one shows that the DL in question has the *tree model property.*
- Second, one devises a translation from pairs $C, \mathcal{T}$, where C is a concept description and $\mathcal{T}$ is a TBox, into an appropriate *tree automata* $\mathcal{A}_{C,\mathcal{T}}$ such that $\mathcal{A}_{C,\mathcal{T}}$ accepts exactly the tree models of C w.r.t. $\mathcal{T}$.
- Third, one applies the *emptiness test* for the employed automaton model to $\mathcal{A}_{C,\mathcal{T}}$ to test whether C has a (tree) model w.r.t. $\mathcal{T}$.

The complexity of the satisfiability algorithm obtained this way depends on the complexity of the translation and the complexity of the emptiness tests. The latter complexity in turn depends on which automaton model is employed.

Below, we will use a simple form of non-deterministic automata working on infinite trees of fixed arity, so-called *looping automata* [116]. In this case, the translation is exponential, but the emptiness test is polynomial (in the size of the already exponentially large automaton obtained through the translation). Thus, the whole algorithm runs in deterministic exponential time. Alternatively, one could use alternating tree automata [92], where a polynomial translation is possible, but the emptiness test is exponential.

Instead of considering automata working on trees of fixed arity, one could also consider so-called amorphous tree automata [26,76], which can deal with arbitrary branching. This simplifies defining the translation, but uses a slightly more complicated automaton model. For some very expressive description logics (e.g., ones that allow for transitive closure of roles [3]), the simple looping automata introduced below are not sufficient since one needs additional acceptance conditions such as the Büchi condition [112] (which requires the occurrence of infinitely many final states in every path).

The Tree Model Property. The first step towards showing that satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes can be decided with the automata-based approach is to establish the tree model property, i.e., to show that any $\mathcal{ALC}$ concept description C satisfiable w.r.t. a general $\mathcal{ALC}$ TBox $\mathcal{T}$ has a tree-shaped model. Note that this model may, in general, be infinite. One way of seeing this is to consider the tableau algorithm introduced above, applied to the ABox $\{C(x)\}$ w.r.t. the representation of the general TBox $\mathcal{T}$ as a single GCI, and just dispose

[15] The cited papers actually use automata-based approaches to show ExpTime results for *extensions* of $\mathcal{ALC}$.

[16] Using the so-called pre-completion technique [64], this is also the case for inference problems involving ABoxes.

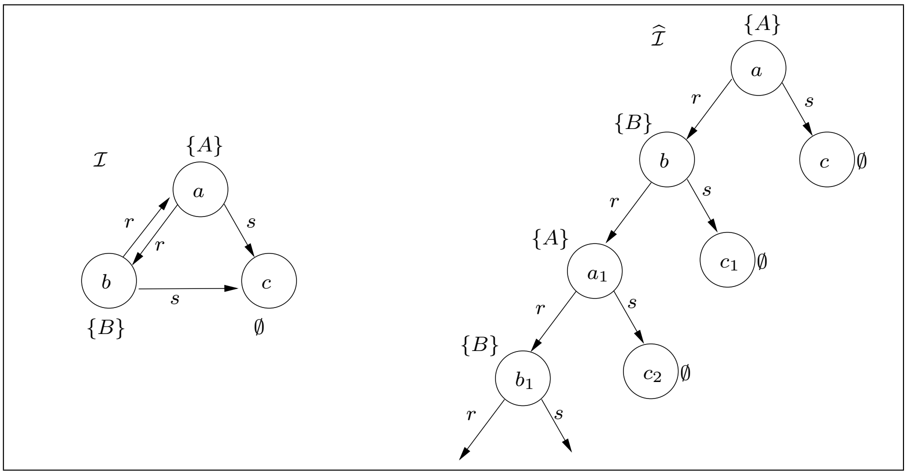


Fig. 7. Unraveling of a model into a tree-shaped model

of blocking. Possibly infinite runs of the algorithm then generate tree-shaped models. However, one can also show the tree model property of $\mathcal{ALC}$ by using the well-known unraveling technique of modal logic [27], in which an arbitrary model of C w.r.t. $\mathcal{T}$ is unraveled into a bisimilar tree-shaped interpretation. Invariance of $\mathcal{ALC}$ under bisimulation [80] (which it inherits from its syntactic variant multimodal K) then implies that the tree shaped interpretation obtained by unraveling is also a model of C w.r.t. $\mathcal{T}$.

Instead of defining unraveling in detail, we just give an example in Fig. 7, and refer the reader to [27] for formal definitions and proofs. The graph on the left-hand side of Fig. 7 describes an interpretation $\mathcal{I}$: the nodes of the graph are the elements of $\Delta^{\mathcal{I}}$, the node labels express to which concept names the corresponding element belongs, and the labelled edges of the graph express the role relationships. For example, $a \in \Delta^{\mathcal{I}}$ belongs to $A^{\mathcal{I}}$, but not to $B^{\mathcal{I}}$, and it has r-successor b and s-successor c. It is easy to check that $\mathcal{I}$ is a model of the concept A w.r.t. the TBox

$$\mathcal{T} := \{A \sqsubseteq \exists r.B, \quad B \sqsubseteq \exists r.A, \quad A \sqcup B \sqsubseteq \exists s.\top\}.$$

The graph on the right-hand side of Fig. 7 describes (a finite part of) the corresponding unraveled model, where a was used as the start node for the unraveling. Basically, one considers all paths starting with a in the original model, but whenever one would re-enter a node one makes a copy of it. Like $\mathcal{I}$, the corresponding unraveled interpretation $\widehat{\mathcal{I}}$ is a model of $\mathcal{T}$ and it satisfies $a \in A^{\widehat{\mathcal{I}}}$.

Looping Tree Automata. As mentioned above, we consider automata working on *infinite trees* of some fixed arity k. To be more precise, the nodes of the trees are labelled by elements from some finite alphabet Σ, whereas the edges are

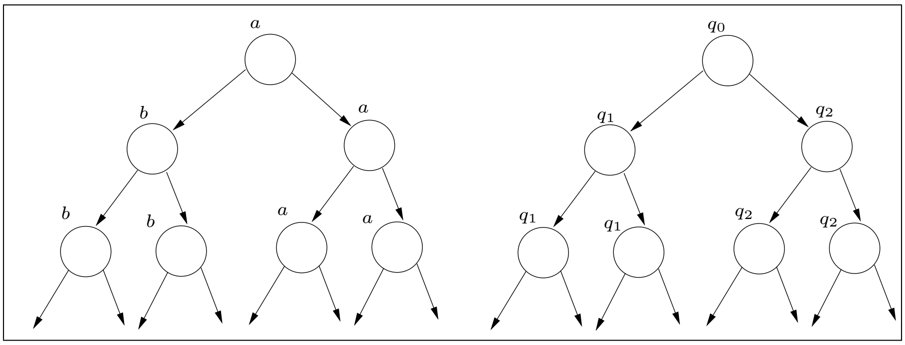


Fig. 8. A tree and a run on it

unlabeled, but ordered, i.e., there is a first, second, to kth successor for each node. Such trees, which we call k-ary Σ-trees, can formally be represented as mappings $T : \{0, \ldots, k-1\}^* \to \Sigma$. Thus, nodes are represented as words over $\{0, \ldots, k-1\}$, the root is the word ε, and a node u has exactly k successor nodes $u0, \ldots, u(k-1)$, and its label is $T(u)$. For example, the binary tree that has root label a, whose left subtree contains only nodes labelled by b, and whose right subtree has only nodes labelled by a (see the left-hand side of Fig. 8) is formally represented as the mapping

$$T : \{0,1\}^* \to \{a,b\} \quad \text{with} \quad T(u) = \begin{cases} b \text{ if } u \text{ starts with } 0 \\ a \text{ otherwise} \end{cases}$$

A *looping automaton* working on k-ary Σ-trees is of the form $\mathcal{A} = (Q, \Sigma, I, \Delta)$, where

- Q is a finite set of states and $I \subseteq Q$ is the set of initial states;
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma \times Q^k$ is the transition relation.

We will usually write tuples $(q, a, q_1, \ldots, q_k) \in \Delta$ in the form $(q, a) \to (q_1, \ldots, q_k)$.

A *run* of $\mathcal{A} = (Q, \Sigma, I, \Delta)$ on the tree $T : \{0, \ldots, k-1\}^* \to \Sigma$ is a k-ary Q-tree $R : \{0, \ldots, k-1\}^* \to Q$ such that $(R(u), T(u)) \to (R(u0), \ldots, R(u(k-1))) \in \Delta$ for all $u \in \{0, \ldots, k-1\}^*$. This run is called *accepting* if $R(\varepsilon) \in I$.

For example, consider the automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$, where

- $Q = \{q_0, q_1, q_2\}$ and $I = \{q_0\}$;
- $\Sigma = \{a, b\}$;
- Δ consists of the transitions

$$(q_0, a) \to (q_1, q_2),\ (q_0, a) \to (q_2, q_1),\ (q_1, b) \to (q_1, q_1),\ (q_2, a) \to (q_2, q_2).$$

The k-ary Q-tree R from the right-hand side of Fig. 8 maps ε to q_0, nodes starting with 0 to q_1, and nodes starting with 1 to q_2. This tree R is an accepting run of $\mathcal{A}$ on the tree T on the left hand side of Figure 8.

The *tree language accepted* by a given looping automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$ is

$$L(\mathcal{A}) := \{T : \{0, \ldots, k-1\}^* \to \Sigma \mid \text{there is an accepting run of } \mathcal{A} \text{ on } T\}.$$

In our example, the language accepted by the automaton consists of two trees, the tree T defined above and the symmetric tree where the left subtree contains only nodes labelled with a and the right subtree contains only nodes labelled with b.

The Emptiness Test. Given a looping tree automaton $\mathcal{A}$, the emptiness test decides whether $L(\mathcal{A}) = \emptyset$ or not. Based on the definition of the accepted language, one might be tempted to try to solve the problem in a *top-down* manner, by first choosing an initial state to label the root, then choosing a transition starting with this state to label its successors, etc. However, the algorithm obtained this way is non-deterministic since one may have several initial states, and also several possible transitions for each state.

To obtain a *deterministic polynomial time emptiness test*, it helps to work *bottom-up*. The main idea is that one wants to compute the set of *bad states*, i.e., states that do not occur in any run of the automaton. Obviously, any state q that does not occur on the left-hand side of a transition $(q, \cdot) \to (\cdots)$ is bad. Starting with this set, one can then extend the set of states known to be bad using the fact that a state q is bad if all transitions $(q, \cdot) \to (q_1, \ldots, q_k)$ starting with q contain a bad state q_j in their right-hand side. Obviously, this process of extending the set of known bad states terminates after a linear number of additions of states to the set of known bad states, and it is easy to show that the final set obtained this way is indeed the set of all bad states. The accepted language is then empty iff all initial states are bad. By using appropriate data structures, one can ensure that the overall complexity of the algorithm is linear in the size of the automaton. A more detailed description of this emptiness test for looping tree automata can be found in [25].

The Reduction. Recall that we want to reduce the satisfiability problem for $\mathcal{ALC}$ concepts w.r.t. general TBoxes to the emptiness problem for looping tree automata by constructing, for a given input $C, \mathcal{T}$, an automaton $\mathcal{A}_{C,\mathcal{T}}$ that accepts exactly the tree-shaped models of C w.r.t. $\mathcal{T}$.

Before this is possible, however, we need to overcome the *mismatch between the underlying kinds of trees*. The tree-shaped models of C w.r.t. $\mathcal{T}$ are trees with labelled edges, but without a fixed arity. In order to express such trees as k-ary Σ-trees for an appropriate k, where Σ consists of all sets of concept names, we consider all the existential restrictions occurring in C and $\mathcal{T}$. The number of these restrictions determines k. Using the bisimulation invariance of $\mathcal{ALC}$ [80], it is easy to show that the existence of a tree-shaped model of C w.r.t. $\mathcal{T}$ also implies the existence of a tree-shaped model where every node has at most k successor nodes. To get exactly k successors, we can do some padding with dummy nodes if needed. The edge label is simply pushed into the label of the successor node, i.e., each node label contains, in addition to concept names, exactly one role name, which expresses with which role the node is reached from its unique predecessor. For the root, an arbitrary role can be chosen.

The states of $\mathcal{A}_{C,\mathcal{T}}$ are sets of subexpressions of the concepts occurring in C and $\mathcal{T}$. Intuitively, a run of the automaton on a tree-shaped model of C w.r.t. $\mathcal{T}$ labels a node not only with the concept names to which this element of the model belongs, but also with all the subexpressions to which it belongs. For technical reasons, we need to normalize the input concept description and TBox before we build these subexpressions. First, we ensure that all GCIs in $\mathcal{T}$ are of the form $\top \sqsubseteq D$ by using the fact that the GCIs $C_1 \sqsubseteq C_2$ and $\top \sqsubseteq \neg C_1 \sqcup C_2$ are equivalent. Second, we transform the input concept description C and every concept D in a GCI $\top \sqsubseteq D$ into negation normal form as described in Section 3.1. In our example, the normalized TBox consists of the GCIs

$$\top \sqsubseteq \neg A \sqcup \exists r.B, \quad \top \sqsubseteq \neg B \sqcup \exists r.A, \quad \top \sqsubseteq (\neg A \sqcap \neg B) \sqcup \exists s.\top,$$

whose subexpressions are $\top, \neg A \sqcup \exists r.B, \neg A, A, \exists r.B, B, \neg B \sqcup \exists r.A, \neg B, \exists r.A, (\neg A \sqcap \neg B) \sqcup \exists s.\top, \neg A \sqcap \neg B, \exists s.\top$. Of these, the node a in the tree-shaped model depicted on the right-hand side of Fig. 7 belongs to $\top, \neg A \sqcup \exists r.B, A, \exists r.B, \neg B \sqcup \exists r.A, \neg B, (\neg A \sqcap \neg B) \sqcup \exists s.\top, \exists s.\top$.

We are now ready to give a *formal definition of the automaton* $\mathcal{A}_{C,\mathcal{T}} = (Q, \Sigma, I, \Delta)$. Let $S_{C,\mathcal{T}}$ denote the set of all subexpressions of C and $\mathcal{T}$, $R_{C,\mathcal{T}}$ denote the set of all role names occurring in C and $\mathcal{T}$, and k the number of existential restrictions contained in $S_{C,\mathcal{T}}$. The *alphabet* Σ basically consists of all subsets of the set of concept names occurring in C and $\mathcal{T}$. As mentioned above, in order to encode the edge labels (i.e., express for which role r the node is a successor node), each "letter" contains, additionally, exactly one role name. Finally, the alphabet contains the empty set (not even containing a role name), which is used to label nodes that are introduced for padding purposes.

The set of *states* Q of $\mathcal{A}_{C,\mathcal{T}}$ consists of the *Hintikka sets* for $C, \mathcal{T}$, i.e., subsets q of $S_{C,\mathcal{T}} \cup R_{C,\mathcal{T}}$ such that $q = \emptyset$ or

– q contains exactly one role name;
– if $\top \sqsubseteq D \in \mathcal{T}$ then $D \in q$;
– if $C_1 \sqcap C_2 \in q$ then $\{C_1, C_2\} \subseteq q$;
– if $C_1 \sqcup C_2 \in q$ then $\{C_1, C_2\} \cap q \neq \emptyset$; and
– $\{A, \neg A\} \not\subseteq q$ for all concept names A.

The set of *initial states* I consists of those states containing C, and the *transition relation* Δ consists of those transitions $(q, \sigma) \rightarrow (q_1, \ldots, q_k)$ satisfying the following properties:

– q and σ coincide w.r.t. the concept and role names contained in them;
– if $q = \emptyset$, then $q_1 = \ldots = q_k = \emptyset$;
– if $\exists r.D \in q$, then there is an i such that $\{D, r\} \subseteq q_i$; and
– if $\forall r.D \in q$ and $r \in q_i$, then $D \in q_i$.

It is not hard to show that the construction of $\mathcal{A}_{C,\mathcal{T}}$ indeed yields a reduction of satisfiability w.r.t. general TBoxes in $\mathcal{ALC}$ to the emptiness problem for looping tree automata.

Proposition 1. *The $\mathcal{ALC}$ concept description C is satisfiable w.r.t. the general $\mathcal{ALC}$ TBox $\mathcal{T}$ iff $L(\mathcal{A}_{C,\mathcal{T}}) \neq \emptyset$.*

Obviously, the number of states of $\mathcal{A}_{C,\mathcal{T}}$ is exponential in the size of C and $\mathcal{T}$. Since the emptiness problem for looping tree automata can be decided in polynomial time, we obtain an deterministic exponential upper-bound for the time complexity of the satisfiability problem. ExpTime-hardness of this problem can be shown by adapting the proof of ExpTime-hardness of satisfiability in propositional dynamic logic (PDL) in [52].

Theorem 2. *Satisfiability in $\mathcal{ALC}$ w.r.t. general TBoxes is* ExpTime*-complete.*

4 Reasoning in the Light-Weight DLs $\mathcal{EL}$ and $\mathcal{FL}_0$

As mentioned in the introduction, early DL systems were based on so-called structural subsumption algorithms, which first normalize the concepts to be tested for subsumption, and then compare the syntactic structure of the normalized concepts. The claim was that these algorithms can decide subsumption in polynomial time. However, the first complexity results for DLs, also mentioned in the introduction, showed that these algorithms were neither polynomial nor decision procedures for subsumption. For example, all early systems used expansion of concept definitions, which can cause an exponential blow-up of the size of concepts. Nebel's coNP-hardness result [94] for subsumption w.r.t. TBoxes showed that this blow-up cannot be avoided whenever the constructors conjunction and value restriction are available. In addition, the early structural subsumption algorithms were not complete, i.e., they were not able to detect all valid subsumption relationships. These negative results for structural subsumption algorithms together with the advent of tableau-based algorithms for expressive DLs, which behaved well in practice, was probably the main reason why structural approaches—and with them the quest for DLs with a polynomial subsumption problem—were largely abandoned during the 1990s. More recent results [6,34,7,8] on the complexity of reasoning in DLs with existential restrictions, rather than value restrictions, have led to a partial rehabilitation of structural approaches and light-weight DLs with polynomial reasoning problems (see the description of *Phase 5* in the introduction).

When trying to find a DL with a polynomial subsumption problem, it is clear that one cannot allow for all Boolean operations, since then one would inherit NP-hardness from propositional logic. It should also be clear that conjunction cannot be dispensed with since one must be able to state that more than one property should hold when defining a concept. Finally, if one wants to call the logic a DL, one needs a constructor using roles. This leads to the following two minimal candidate DLs:

- the DL $\mathcal{FL}_0$ [4], which offers the concept constructors conjunction, value restriction ($\forall r.C$), and the top concept;
- the DL $\mathcal{EL}$ [7], which offers the concept constructors conjunction, existential restriction ($\exists r.C$), and the top concept.

In the following, we will look at the subsumption problem[17] in these two DLs in some detail. Whereas subsumption without a TBox turns out to be polynomial in both cases, we will also see that $\mathcal{EL}$ exhibits a more robust behavior w.r.t. the complexity of the subsumption problem in the presence of TBoxes.

Subsumption in $\mathcal{FL}_0$. First, we consider the case of subsumption of $\mathcal{FL}_0$-concept descriptions without a TBox. There are basically two approaches for obtaining a structural subsumption algorithm in this case, which are based on two different normal forms. One can either use the equivalence $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as a rewrite rule from left-to-right or from right-to-left. Here we will consider the approach based on the left-to-right direction, whereas all of the early structural subsumption algorithms were based on a normal form obtained by rewriting in the other direction.[18]

By using the rewrite rule $\forall r.(C \sqcap D) \rightarrow \forall r.C \sqcap \forall r.D$ together with associativity, commutativity and idempotence[19] of $\sqcap$, any $\mathcal{FL}_0$-concept can be transformed into an equivalent one that is a conjunction of concepts of the form $\forall r_1.\cdots\forall r_m.A$ for $m \geq 0$ (not necessarily distinct) role names $r_1, \ldots, r_m$ and a concept name A. We abbreviate $\forall r_1.\cdots\forall r_m.A$ by $\forall r_1 \ldots r_m.A$, where $r_1 \ldots r_m$ is viewed as a word over the alphabet of all role names. In addition, instead of $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ we write $\forall L.A$ where $L := \{w_1, \ldots, w_\ell\}$ is a finite set of words over Σ. The term $\forall\emptyset.A$ is considered to be equivalent to the top concept $\top$, which means that it can be added to a conjunction without changing the meaning of the concept. Using these abbreviations, any pair of $\mathcal{FL}_0$-concept descriptions C, D containing the concept names $A_1, \ldots, A_k$ can be rewritten as

$$C \equiv \forall U_1.A_1 \sqcap \ldots \sqcap \forall U_k.A_k \quad \text{and} \quad D \equiv \forall V_1.A_1 \sqcap \ldots \sqcap \forall V_k.A_k,$$

where U_i, V_i are finite sets of words over the alphabet of all role names. This normal form provides us with the following *characterization of subsumption* of $\mathcal{FL}_0$-concept descriptions [20]:

$$C \sqsubseteq D \quad \text{iff} \quad U_i \supseteq V_i \quad \text{for all } i, 1 \leq i \leq k.$$

Since the size of the normal forms is polynomial in the size of the original concept descriptions, and since the inclusion tests $U_i \supseteq V_i$ can also be realized in polynomial time, this yields a polynomial-time decision procedure for subsumption in $\mathcal{FL}_0$.

This characterization of subsumption via inclusion of finite sets of words can be extended to TBoxes as follows. A given TBox $\mathcal{T}$ can be translated into a finite (word) automaton[20] $\mathcal{A}_\mathcal{T}$, whose states are the concept names occurring in

[17] Note that the satisfiability problem is trivial in $\mathcal{FL}_0$ and $\mathcal{EL}$, since any concept expressed in these languages is satisfiable. The reduction of subsumption to satisfiability is not possible due to the absence of negation.

[18] A comparison between the two approaches can be found in [17].

[19] I.e., $(A \sqcap B) \sqcap C \equiv A \sqcap (B \sqcap C)$, $A \sqcap B \equiv B \sqcap A$, and $A \sqcap A \equiv A$.

[20] Strictly speaking, we obtain a finite automaton with word transitions, i.e., transitions that may be labelled with a word over Σ rather than a letter of Σ.

$$A \equiv C \sqcap \forall r.B \sqcap \forall s.\forall r.P$$
$$B \equiv \forall s.C$$
$$C \equiv \forall r.P$$

$$\begin{tikzcd}
B \arrow[r, "s"] & C \arrow[dr, "r"] & \\
A \arrow[u, "r"] \arrow[ur, "\varepsilon"'] \arrow[rr, "sr"'] & & P
\end{tikzcd}$$

Fig. 9. An $\mathcal{FL}_0$ TBox and the corresponding acyclic automaton

$\mathcal{T}$, and whose transitions are induced by the value restrictions occurring in $\mathcal{T}$ (see Fig. 9 for an example). A formal definition of this translation can be found in [4], where the more general case of cyclic TBoxes is treated. In the case of TBoxes, which are by definition acyclic, the resulting automata are also acyclic.

For a defined concept A and a primitive concept P in $\mathcal{T}$, the language $L_{\mathcal{A}_\mathcal{T}}(A, P)$ is the set of all words labeling paths in $\mathcal{A}_\mathcal{T}$ from A to P. The languages $L_{\mathcal{A}_\mathcal{T}}(A, P)$ represent all the value restrictions that must be satisfied by instances of the concept A. With this intuition in mind, it should not be surprising that subsumption w.r.t. $\mathcal{FL}_0$ TBoxes can be characterized in terms of inclusion of languages accepted by acyclic automata. Indeed, the following is a *characterization of subsumption* in $\mathcal{FL}_0$ w.r.t. TBoxes:

$$A \sqsubseteq_\mathcal{T} B \quad \text{iff} \quad L_{\mathcal{A}_\mathcal{T}}(A, P) \supseteq L_\mathcal{T}(B, P) \;\; \text{for all primitive concepts } P.$$

In the example of Fig. 9, we have $L_{\mathcal{A}_\mathcal{T}}(A, P) = \{r, sr, rsr\} \supset \{sr\} = L_{\mathcal{A}_\mathcal{T}}(B, P)$, and thus $A \sqsubseteq_\mathcal{T} B$, but $B \not\sqsubseteq_\mathcal{T} A$.

Since the inclusion problem for languages accepted by acyclic finite automata is coNP-complete [54], this reduction shows that the subsumption problem in $\mathcal{FL}_0$ w.r.t. TBoxes is in coNP. As shown by Nebel [94], the reduction also works in the opposite direction, which yields the matching lower bound. For cyclic TBoxes, the subsumption problem corresponds to the inclusion problem for languages accepted by arbitrary finite automata, which is PSpace-complete, and thus the subsumption problem is also PSpace-complete [4,77]. In the presence of general TBoxes, the subsumption problem in $\mathcal{FL}_0$ actually becomes as hard as for $\mathcal{ALC}$, namely ExpTime-hard [7,63].

Theorem 3. *Subsumption in $\mathcal{FL}_0$ is polynomial without TBox, coNP-complete w.r.t. TBoxes, PSpace-complete w.r.t. cyclic TBoxes, and* ExpTime-*complete w.r.t. general TBoxes.*

Subsumption in $\mathcal{EL}$. In contrast to the negative complexity results for subsumption w.r.t. TBoxes in $\mathcal{FL}_0$, subsumption in $\mathcal{EL}$ remains polynomial even in the presence of general TBoxes [34].[21] The polynomial-time subsumption algorithm for $\mathcal{EL}$ that will be sketched below actually classifies a given TBox $\mathcal{T}$, i.e.,

[21] The special case of cyclic TBoxes was already treated in [6].

it simultaneously computes all subsumption relationships between the concept names occurring in $\mathcal{T}$. This algorithm proceeds in four steps:

1. Normalize the TBox.
2. Translate the normalized TBox into a graph.
3. Complete the graph using completion rules.
4. Read off the subsumption relationships from the normalized graph.

A general $\mathcal{EL}$-TBox is *normalized* if it only contains GCIs of the following form:

$$A_1 \sqcap A_2 \sqsubseteq B, \quad A \sqsubseteq \exists r.B, \quad \text{or} \quad \exists r.A \sqsubseteq B,$$

where A, A_1, A_2, B are concept names or the top-concept $\top$. One can transform a given TBox into a normalized one by applying normalization rules. Instead of describing these rules in the general case, we just illustrate them by an example, where we underline GCIs on the right-hand side that need further rewriting:

$$\begin{array}{rcl}
\exists r.A \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B & \rightsquigarrow & \exists r.A \sqsubseteq B_1,\ \underline{B_1 \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B} \\
B_1 \sqcap \exists r.\exists s.A \sqsubseteq A \sqcap B & \rightsquigarrow & \underline{\exists r.\exists s.A \sqsubseteq B_2},\ \underline{B_1 \sqcap B_2 \sqsubseteq A \sqcap B} \\
\exists r.\exists s.A \sqsubseteq B_2 & \rightsquigarrow & \exists s.A \sqsubseteq B_3,\ \exists r.B_3 \sqsubseteq B_2, \\
B_1 \sqcap B_2 \sqsubseteq A \sqcap B & \rightsquigarrow & B_1 \sqcap B_2 \sqsubseteq A,\ B_1 \sqcap B_2 \sqsubseteq B
\end{array}$$

For example, in the first normalization step we introduce the abbreviation B_1 for the description $\exists r.A$. One might think that one must make B_1 equivalent to $\exists r.A$, i.e., also add the GCI $B_1 \sqsubseteq \exists r.A$. However, it can be shown that adding just $\exists r.A \sqsubseteq B_1$ is sufficient to obtain a *subsumption-equivalent* TBox, i.e., a TBox that induces the same subsumption relationships between the concept names occurring in the original TBox. All normalization rules preserve equivalence in this sense, and if one uses an appropriate strategy (which basically defers the applications of the rule applied in the last step of our example to the end), then the normal form can be computed by a linear number of rule applications.

In the next step, we build the *classification graph* $G_{\mathcal{T}} = (V, V \times V, S, R)$ where

– V is the set of concept names (including $\top$) occurring in the normalized TBox $\mathcal{T}$;
– S labels nodes with sets of concept names (again including $\top$);
– R labels edges with sets of role names.

It can be shown that the label sets satisfy the following *invariants*:

– $B \in S(A)$ implies $A \sqsubseteq_{\mathcal{T}} B$, i.e., $S(A)$ contains only subsumers of A w.r.t. $\mathcal{T}$.
– $r \in R(A, B)$ implies $A \sqsubseteq_{\mathcal{T}} \exists r.B$, i.e., $R(A, B)$ contains only roles r such that $\exists r.B$ subsumes A w.r.t. $\mathcal{T}$.

Initially, we set $S(A) := \{A, \top\}$ for all nodes $A \in V$, and $R(A, B) := \emptyset$ for all edges $(A, B) \in V \times V$. Obviously, the above invariants are satisfied by these initial label sets.

(R1)	$A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$	and	$A_1, A_2 \in S(A)$	then	add B to $S(A)$
(R2)	$A_1 \sqsubseteq \exists r.B \in \mathcal{T}$	and	$A_1 \in S(A)$	then	add r to $R(A, B)$
(R3)	$\exists r.B_1 \sqsubseteq A_1 \in \mathcal{T}$	and	$B_1 \in S(B), r \in R(A, B)$	then	add A_1 to $S(A)$

Fig. 10. The completion rules for subsumption in $\mathcal{EL}$ w.r.t. general TBoxes

The labels of nodes and edges are then extended by applying the rules of Fig. 10, where we assume that a rule is only applied if it really extends a label set. It is easy to see that these rules preserve the above invariants. For example, consider the (most complicated) rule (R3). Obviously, $\exists r.B_1 \sqsubseteq A_1 \in \mathcal{T}$ implies $\exists r.B_1 \sqsubseteq_{\mathcal{T}} A_1$, and the assumption that the invariants are satisfied before applying the rule yields $B \sqsubseteq_{\mathcal{T}} B_1$ and $A \sqsubseteq_{\mathcal{T}} \exists r.B$. The subsumption relationship $B \sqsubseteq_{\mathcal{T}} B_1$ obviously implies $\exists r.B \sqsubseteq_{\mathcal{T}} \exists r.B_1$. By applying transitivity of the subsumption relation $\sqsubseteq_{\mathcal{T}}$, we thus obtain $A \sqsubseteq_{\mathcal{T}} A_1$.

The fact that subsumption in $\mathcal{EL}$ w.r.t. general TBoxes can be decided in polynomial time is an immediate consequence of the following statements:

1. Rule application terminates after a polynomial number of steps.
2. If no more rules are applicable, then $A \sqsubseteq_{\mathcal{T}} B$ iff $B \in S(A)$.

Regarding the first statement, note that the number of nodes is linear and the number of edges is quadratic in the size of $\mathcal{T}$. In addition, the size of the label sets is bounded by the number of concept names and role names, and each rule application extends at least one label. Regarding the equivalence in the second statement, the "if" direction follows from the fact that the above invariants are preserved under rule application. To show the "only-if" direction, assume that $B \notin S(A)$. Then the following interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ in which $A \in A^{\mathcal{I}}$, but $A \notin B^{\mathcal{I}}$:

– $\Delta^{\mathcal{I}} := V$;
– $r^{\mathcal{I}} := \{(A', B') \mid r \in R(A', B')\}$ for all role names r;
– $B'^{\mathcal{I}} := \{A' \mid B' \in S(A')\}$ for all concept names A'.

More details can be found in [34,7].

Theorem 4. *Subsumption in $\mathcal{EL}$ is polynomial w.r.t. general TBoxes.*

In [7] this result is extended to the DL $\mathcal{EL}^{++}$, which extends $\mathcal{EL}$ with the bottom concept, nominals, a restricted form of concrete domains, and a restricted form of so-called role-value maps. In addition, it is shown in [7] that almost all additions of other typical DL constructors to $\mathcal{EL}$ make subsumption w.r.t. general TBoxes ExpTime-complete.

It should be noted that these results are not only of theoretical interest. In fact, both the large medical ontology Snomed ct[22] and the Gene Ontology[23]

[22] http://www.ihtsdo.org/snomed-ct/
[23] http://www.geneontology.org/

can be expressed in $\mathcal{EL}$, and the same is true for large parts of the medical ontology GALEN [102]. First implementations of the subsumption algorithm for $\mathcal{EL}$ sketched above behave well on these very large knowledge bases [19,81,111].

In [8], the DL $\mathcal{EL}^{++}$ is extended with reflexive roles and range restrictions since these means of expressivity have turned out to be important in medical ontologies. It is shown that subsumption remains tractable if a certain syntactic restriction is adopted. The DL obtained this way corresponds closely to the OWL 2 profile OWL 2 EL.[24]

Acknowledgement

This article is, on the one hand, based on the Description Logic tutorial by the author, which he first taught at the 2005 Logic Summer School organized by the Research School of Information Sciences and Engineering, Australian National University, Canberra, Australia. This tutorial in turn took some inspirations from the Description Logic tutorial taught by Carsten Lutz and Ulrike Sattler at the 2005 ICCL Summer School on Logic-Based Knowledge Representation organized by the International Center for Computational Logic, TU Dresden, Germany. On the other hand, this article reuses some of the material from the overview articles [23,18,16], written by the author in collaboration with Ian Horrocks, Carsten Lutz, and Ulrike Sattler.

References

1. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: Quonto: Querying ontologies. In: Veloso, M.M., Kambhampati, S. (eds.) Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 1670–1671. AAAI Press/The MIT Press (2005)
2. Areces, C., de Rijke, M., de Nivelle, H.: Resolution in modal, description and hybrid logic. J. of Logic and Computation 11(5), 717–736 (2001)
3. Baader., F.: Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence, IJCAI 1991 (1991)
4. Baader, F.: Using automata theory for characterizing the semantics of terminological cycles. Ann. of Mathematics and Artificial Intelligence 18, 175–219 (1996)
5. Baader, F.: Description logic terminology. In: [11], pp. 485–495 (2003)
6. Baader, F.: Terminological cycles in a description logic with existential restrictions. In: Gottlob, G., Walsh, T. (eds.) Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp. 325–330. Morgan Kaufmann, Los Altos (2003)
7. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), Edinburgh, UK, pp. 364–369. Morgan Kaufmann, Los Altos (2005)
8. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Clark, K., Patel-Schneider, P.F. (eds.) Proceedings of the Fifth International Workshop on OWL: Experiences and Directions (OWLED 2008), Karlsruhe, Germany (2008)

[24] http://www.w3.org/TR/owl2-profiles/

9. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. Artificial Intelligence 88(1–2), 195–213 (1996)
10. Baader, F., Bürckert, H.-J., Nebel, B., Nutt, W., Smolka, G.: On the expressivity of feature logics with negation, functional uncertainty, and sort equations. J. of Logic, Language and Information 2, 1–18 (1993)
11. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
12. Baader, F., Franconi, E., Hollunder, B., Nebel, B., Profitlich, H.-J.: An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. Applied Artificial Intelligence. Special Issue on Knowledge Base Management 4, 109–132 (1994)
13. Baader, F., Hladik, J., Lutz, C., Wolter, F.: From tableaux to automata for description logics. Fundamenta Informaticae 57(2–4), 247–279 (2003)
14. Baader, F., Hollunder, B.: A terminological knowledge representation system with complete inference algorithm. In: Boley, H., Richter, M.M. (eds.) PDK 1991. LNCS (LNAI), vol. 567, pp. 67–86. Springer, Heidelberg (1991)
15. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. International Handbooks in Information Systems, pp. 3–28. Springer, Berlin (2003)
16. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 135–179. Elsevier, Amsterdam (2007)
17. Baader, F., Küsters, R., Molitor, R.: Structural subsumption considered from an automata theoretic point of view. In: Proc. of the 1998 Description Logic Workshop (DL 1998). CEUR Electronic Workshop Proceedings (1998), `http://ceur-ws.org/Vol-11/`
18. Baader, F., Lutz, C.: Description logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) The Handbook of Modal Logic, pp. 757–820. Elsevier, Amsterdam (2006)
19. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—a polynomial-time reasoner for life science ontologies. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 287–291. Springer, Heidelberg (2006)
20. Baader, F., Narendran, P.: Unification of concepts terms in description logics. J. of Symbolic Computation 31(3), 277–305 (2001)
21. Baader, F., Nutt, W.: Basic description logics. In: [11], pp. 43–95 (2003)
22. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS, vol. 4667, pp. 52–67. Springer, Heidelberg (2007)
23. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69, 5–40 (2001)
24. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED 2008), Phoenix, Arizona (2008)
25. Baader, F., Tobies, S.: The inverse method implements the automata approach for modal satisfiability. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 92–106. Springer, Heidelberg (2001)
26. Bernholtz, O., Grumberg, O.: Branching time temporal logic and amorphous tree automata. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 262–277. Springer, Heidelberg (1993)

27. Blackburn, P., de Rijke, M., de Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
28. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82(1–2), 353–367 (1996)
29. Brachman, R.J.: "Reducing" CLASSIC to practice: Knowledge representation meets reality. In: Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1992), pp. 247–258. Morgan Kaufmann, Los Altos (1992)
30. Brachman, R.J., Levesque, H.J.: The tractability of subsumption in frame-based description languages. In: Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI 1984), pp. 34–37 (1984)
31. Brachman, R.J., Levesque, H.J.: Readings in Knowledge Representation. Morgan Kaufmann, Los Altos (1985)
32. Brachman, R.J., Nardi, D.: An introduction to description logics. In: [11], pp. 1–40 (2003)
33. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2), 171–216 (1985)
34. Brandt., S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In: de Mántaras, R.L., Saitta, L. (eds.) Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004), pp. 298–302 (2004)
35. Bresciani, P., Franconi, E., Tessaris, S.: Implementing and testing expressive description logics: Preliminary report. In: Proc. of the 1995 Description Logic Workshop (DL 1995), pp. 131–139 (1995)
36. Buchheit, M., Donini, F.M., Nutt, W., Schaerf, A.: A refined architecture for terminological systems: Terminology = schema + views. Artificial Intelligence 99(2), 209–260 (1998)
37. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. J. of Artificial Intelligence Research 1, 109–138 (1993)
38. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Veloso, M.M., Kambhampati, S. (eds.) Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 602–607. AAAI Press/The MIT Press (2005)
39. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning 39(3), 385–429 (2007)
40. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1998), pp. 149–158 (1998)
41. Calvanese, D., De Giacomo, G., Lenzerini, M.: Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 1999), pp. 84–89 (1999)
42. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description logic framework for information integration. In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 1998), pp. 2–13 (1998)

43. De Giacomo, G.: Decidability of Class-Based Knowledge Representation Formalisms. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza" (1995)
44. De Giacomo, G., Lenzerini, M.: Boosting the correspondence between description logics and propositional dynamic logics. In: Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI 1994), pp. 205–212. AAAI Press/The MIT Press (1994)
45. De Giacomo, G., Lenzerini, M.: Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In: Proc. of the 11th Eur. Conf. on Artificial Intelligence (ECAI 1994), pp. 411–415 (1994)
46. De Giacomo, G., Lenzerini, M.: TBox and ABox reasoning in expressive description logics. In: Aiello, L.C., Doyle, J., Shapiro, S.C. (eds.) ECAI-WS 1992, pp. 316–327. Morgan Kaufmann, Los Altos (1996)
47. Donini, F.: Complexity of reasoning. In: [11], pp. 96–136 (2003)
48. Donini, F., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. Acta Informatica 124(1), 87–138 (2000)
49. Donini, F.M., Hollunder, B., Lenzerini, M., Spaccamela, A.M., Nardi, D., Nutt, W.: The complexity of existential quantification in concept languages. Artificial Intelligence 2–3, 309–327 (1992)
50. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: The complexity of concept languages. In: Allen, J., Fikes, R., Sandewall, E. (eds.) Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1991), pp. 151–162. Morgan Kaufmann, Los Altos (1991)
51. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: Tractable concept languages. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991), Sydney, Australia, pp. 458–463 (1991)
52. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. of Computer and System Sciences 18, 194–211 (1979)
53. Fitting, M.: Tableau methods of proof for modal logics. Notre Dame J. of Formal Logic 13(2), 237–247 (1972)
54. Garey, M.R., Johnson, D.S.: Computers and Intractability — A guide to NP-completeness. W. H. Freeman and Company, San Francisco (1979)
55. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In: Veloso, M.M. (ed.) Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), Hyderabad, India, pp. 399–404 (2007)
56. Goré, R., Nguyen, L.A.: Exptime tableaux for $\mathcal{ALC}$ using sound global caching. In: Proc. of the 2007 Description Logic Workshop (DL 2007), Brixen-Bressanone, Italy (2007)
57. Grädel, E.: Guarded fragments of first-order logic: A perspective for new description logics? In: Proc. of the 1998 Description Logic Workshop (DL 1998). CEUR Electronic Workshop Proceedings (1998), http://ceur-ws.org/Vol-11/
58. Grädel, E.: On the restraining power of guards. J. of Symbolic Logic 64, 1719–1742 (1999)
59. Grädel, E., Kolaitis, P.G., Vardi, M.Y.: On the decision problem for two-variable first-order logic. Bulletin of Symbolic Logic 3(1), 53–69 (1997)
60. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: A logical framework for modularity of ontologies. In: Veloso, M.M. (ed.) Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), Hyderabad, India, pp. 298–303 (2007)
61. Haarslev, V., Möller, R.: RACE system description. In: Proc. of the 1999 Description Logic Workshop (DL 1999). CEUR Electronic Workshop Proceedings, pp. 130–132 (1999), http://ceur-ws.org/Vol-22/

62. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–706. Springer, Heidelberg (2001)
63. Hofmann, M.: Proof-theoretic approach to description-logic. In: Panangaden, P. (ed.) Proc. of the 20th IEEE Symp. on Logic in Computer Science (LICS 2005), pp. 229–237. IEEE Computer Society Press, Los Alamitos (2005)
64. Hollunder, B.: Consistency checking reduced to satisfiability of concepts in terminological systems. Ann. of Mathematics and Artificial Intelligence 18(2–4), 133–157 (1996)
65. Hollunder, B., Baader, F.: Qualifying number restrictions in concept languages. In: Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1991), pp. 335–346 (1991)
66. Hollunder, B., Nutt, W., Schmidt-Schauß, M.: Subsumption algorithms for concept description languages. In: Proc. of the 9th Eur. Conf. on Artificial Intelligence (ECAI 1990), London, United Kingdom, Pitman, pp. 348–353 (1990)
67. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 1998), pp. 636–647 (1998)
68. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District, UK, pp. 57–67. AAAI Press/The MIT Press (2006)
69. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
70. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of Logic and Computation 9(3), 385–410 (1999)
71. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS (LNAI), vol. 1705, pp. 161–180. Springer, Heidelberg (1999)
72. Hustadt, U., Motik, B., Sattler, U.: Reducing SHIQ-description logic to disjunctive datalog programs. In: Dubois, D., Welty, C.A., Williams, M.-A. (eds.) Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004), pp. 152–162. Morgan Kaufmann, Los Altos (2004)
73. Hustadt, U., Schmidt., R.A.: On the relation of resolution and tableaux proof systems for description logics. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 1999), pp. 110–117 (1999)
74. Hustadt, U., Schmidt, R.A.: Issues of decidability for description logics in the framework of resolution. In: Caferra, R., Salzer, G. (eds.) FTP 1998. LNCS (LNAI), vol. 1761, pp. 191–205. Springer, Heidelberg (2000)
75. Hustadt, U., Schmidt, R.A., Georgieva, L.: A survey of decidable first-order fragments and description logics. Journal of Relational Methods in Computer Science 1, 251–276 (2004)
76. Janin, D., Walukiewicz, I.: Automata for the modal mu-calculus and related results. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 552–562. Springer, Heidelberg (1995)
77. Kazakov, Y., de Nivelle, H.: Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In: Proc. of the 2003 Description Logic Workshop (DL 2003). CEUR Electronic Workshop Proceedings (2003), `http://CEUR-WS.org/Vol-81/`

78. Kazakov, Y., Motik, B.: A resolution-based decision procedure for $\mathcal{SHOIQ}$. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 662–677. Springer, Heidelberg (2006)
79. Konev, B., Lutz, C., Walther, D., Wolter, F.: Semantic modularity and module extraction in description logics. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N. (eds.) Proc. of the 18th Eur. Conf. on Artificial Intelligence (ECAI 2008), pp. 55–59. IOS Press, Amsterdam (2008)
80. Kurtonina, N., de Rijke, M.: Expressiveness of concept expressions in first-order description logics. Artificial Intelligence 107(2), 303–333 (1999)
81. Lawley., M.: Exploiting fast classification of SNOMED CT for query and integration of health data. In: Cornet, R., Spackman, K. (eds.) Proc. of the 3rd Int. Conf. on Knowledge Representation in Medicine (KR-MED 2008), Phoenix, Arizona, USA (2008)
82. Levesque, H.J., Brachman, R.J.: Expressiveness and tractability in knowledge representation and reasoning. Computational Intelligence 3, 78–93 (1987)
83. Lutz., C.: Complexity of terminological reasoning revisited. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS (LNAI), vol. 1705, pp. 181–200. Springer, Heidelberg (1999)
84. Lutz, C.: Interval-based temporal reasoning with general TBoxes. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), pp. 89–94 (2001)
85. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 179–193. Springer, Heidelberg (2008)
86. Lutz, C., Sattler, U.: Mary likes all cats. In: Proc. of the 2000 Description Logic Workshop (DL 2000). CEUR Electronic Workshop Proceedings, pp. 213–226 (2000), http://ceur-ws.org/Vol-33/
87. MacGregor, R.: The evolving technology of classification-based knowledge representation systems. In: Sowa, J.F. (ed.) Principles of Semantic Networks, pp. 385–400. Morgan Kaufmann, Los Altos (1991)
88. Mays, E., Dionne, R., Weida, R.: K-REP system overview. SIGART Bull. 2(3) (1991)
89. Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), AAAI Press/The MIT Press (2006)
90. Minsky, M.: A framework for representing knowledge. In: Haugeland, J. (ed.) Mind Design. The MIT Press, Cambridge (1981); A longer version appeared in The Psychology of Computer Vision (1975), Republished in [31]
91. Mortimer, M.: On languages with two variables. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 21, 135–140 (1975)
92. Muller, D.E., Schupp, P.E.: Alternating automata on infinite trees. Theoretical Computer Science 54, 267–276 (1987)
93. Nebel, B.: Reasoning and Revision in Hybrid Representation Systems. LNCS (LNAI), vol. 422. Springer, Heidelberg (1990)
94. Nebel, B.: Terminological reasoning is inherently intractable. Artificial Intelligence 43, 235–249 (1990)
95. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. of Automated Reasoning 41(1), 61–98 (2008)
96. Pacholski, L., Szwast, W., Tendera, L.: Complexity of two-variable logic with counting. In: Proc. of the 12th IEEE Symp. on Logic in Computer Science (LICS 1997), pp. 318–327. IEEE Computer Society Press, Los Alamitos (1997)

97. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Ellis, A., Hagino, T. (eds.) Proc. of the 14th International Conference on World Wide Web (WWW 2005), pp. 633–640. ACM, New York (2005)
98. Patel-Schneider, P.F.: DLP. In: Proc. of the 1999 Description Logic Workshop (DL 1999). CEUR Electronic Workshop Proceedings, pp. 9–13 (1999), http://ceur-ws.org/Vol-22/
99. Patel-Schneider, P.F., McGuiness, D.L., Brachman, R.J., Resnick, L.A., Borgida, A.: The CLASSIC knowledge representation system: Guiding principles and implementation rational. SIGART Bull. 2(3), 108–113 (1991)
100. Peltason, C.: The BACK system — an overview. SIGART Bull. 2(3), 114–119 (1991)
101. Ross Quillian, M.: Semantic memory. In: Minsky, M. (ed.) Semantic Information Processing, pp. 216–270. The MIT Press, Cambridge (1968)
102. Rector, A., Horrocks, I.: Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In: Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI 1997), Stanford, CA. AAAI Press, Menlo Park (1997)
103. Schild., K.: A correspondence theory for terminological logics: Preliminary report. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991), pp. 466–471 (1991)
104. Schild, K.: Querying Knowledge and Data Bases by a Universal Description Logic with Recursion. PhD thesis, Universität des Saarlandes, Germany (1995)
105. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp. 355–362. Morgan Kaufmann, Los Altos (2003)
106. Schmidt-Schauß, M.: Subsumption in KL-ONE is undecidable. In: Brachman, R.J., Levesque, H.J., Reiter, R. (eds.) Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1989), pp. 421–431. Morgan Kaufmann, Los Altos (1989)
107. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with unions and complements. Technical Report SR-88-21, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany (1988)
108. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
109. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Proc. of the 2004 Description Logic Workshop (DL 2004), pp. 212–213 (2004)
110. Suntisrivaraporn, B.: Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}^+$ ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 230–244. Springer, Heidelberg (2008)
111. Suntisrivaraporn, B.: Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies. PhD thesis, Fakultät Informatik, TU Dresden (2009), http://lat.inf.tu-dresden.de/research/phd/#Sun-PhD-2008
112. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, ch. 4, vol. B, pp. 134–189. Elsevier Science Publishers, Amsterdam (1990)

113. Tobies, S.: A PSPACE algorithm for graded modal logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 52–66. Springer, Heidelberg (1999)
114. Tobies, S.: Complexity Results and Practical Algorithms for Logics in Knowledge Representation. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany (2001)
115. Tsarkov, D., Horrocks, I.: faCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)
116. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)

Answer Set Programming: A Primer*

Thomas Eiter[1], Giovambattista Ianni[2], and Thomas Krennwallner[1]

[1] Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,tkren}@kr.tuwien.ac.at
[2] Dipartimento di Matematica, Universitá della Calabria, I-87036 Rende (CS), Italy
ianni@mat.unical.it

Abstract. Answer Set Programming (ASP) is a declarative problem solving paradigm, rooted in Logic Programming and Nonmonotonic Reasoning, which has been gaining increasing attention during the last years. This article is a gentle introduction to the subject; it starts with motivation and follows the historical development of the challenge of defining a semantics for logic programs with negation. It looks into positive programs over stratified programs to arbitrary programs, and then proceeds to extensions with two kinds of negation (named weak and strong negation), and disjunction in rule heads. The second part then considers the ASP paradigm itself, and describes the basic idea. It shows some programming techniques and briefly overviews Answer Set solvers. The third part is devoted to ASP in the context of the Semantic Web, presenting some formalisms and mentioning some applications in this area. The article concludes with issues of current and future ASP research.

1 Introduction

Over the the last years, *Answer Set Programming* (ASP) [108,62,82,92,96] has emerged as a declarative problem solving paradigm that has its roots in Logic Programming and Nonmonotonic Reasoning. This particular way of programming, in a language which is sometimes called AnsProlog (or simply A-Prolog) [10,60], is well-suited for modeling and (automatically) solving problems which involve common sense reasoning: it has been fruitfully applied to a range of applications (for more details, see Section 6). A number of extensions of the ASP core language, which goes back to the seminal paper by Gelfond and Lifschitz [63], have been developed (resulting in an AnsProlog* language family). These extensions aim at increasing the expressiveness of the formalisms and/or providing convenient constructs for application-specific problem representation; see, e.g., [97] for an account of such extensions.

The basic idea of ASP is to describe problem specifications by means of a nonmonotonic logic program: solutions to instances of such a problem will be represented by the intended models of the program (the so-called *answer sets*, or *stable models*)

* This work has been supported by the Austrian Science Fund (FWF) project P20840 & P20841, the EC ICT Integrated Project Ontorule (FP7 231875), and the Italian National Project Interlink II04CG8AGG.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Fig. 1. Sudoku puzzle (left) and solution (right)

at hand. Rules and constraints, which *describe* the problem and its possible solutions rather than a concrete algorithm, are basic elements of such programs.

Such a problem encoding can be then fed into an answer set (AS) solver, which computes some or multiple answer set(s) of the program, from which the solutions of the problem can easily be read off.

As a simple motivating example, consider the popular Sudoku game.[1]

Example 1 (Sudoku). In its original version, a Sudoku consists of a tableau that has 81 cells arranged in a grid, which is divided into nine sub-tableaux (the blocks or regions) of equal size having nine fields each. The initial game setup has some of the entries filled with numbers between 1 and 9 (see Figure 1, left, for an example).

The question is now whether the tableau can be completed in a way such that each row and each column shows every digits from 1 to 9 exactly once, and moreover that also each block has this property. An example for a completed Sudoku grid is on the right in Figure 1, which is the unique solution to the initial puzzle on the left.[2]

In general, the problem of solving Sudoku tables automatically appears to be non-trivial: in principle, one can devise a brute force algorithm that considers all possible assignments and checks whether the solution constraint is satisfied. For a versatile programmer, it is not difficult to write a program in her favorite programming language, be it Java, C++, or some other language, to compute and print a solution to instances of this problem.

In this traditional, time-consuming approach, a human programmer receives an informal specification of the problem at hand, such as the Sudoku above, and manually converts it into imperative code that is able to solve instances of the problem. However, one might conceive to tackle this issue from a completely different perspective.

For instance, one can think of having access to appropriate means for *directly describing* the problem at hand in a declarative specification. This specification, if properly polished from ambiguities of natural language and expressed in a proper syntax, would be not much different in its meaning from the formulation of Sudoku of our

[1] This game has nowadays worldwide popularity, and world and national championships are held in big tournaments each year across Europe.

[2] To date, many variants of Sudoku emerged, like, e.g., color-Sudoku, Samurai-Sudoku, etc.

example. Also, such a specification could be automatically *executed*, in the sense that some computational engine *takes this specification as input*, together with a problem instance, and then *produces a solution as output*. In such a vision, the human programmer would switch her focus from *how to solve a problem* to *how to state a problem*, which is a much easier and faster task.[3]

The Prolog language, and its extensions conceived for handling constraints, can be seen at a first glance as tools for such "declarative problem solving." Prolog is indeed well-suited for this particular case.

There are however aspects which make the suitability of Prolog (with respect to AnsProlog) less apparent. Among such aspects, there is the fact that many common problems require preference handling (that is, the possibility to describe which solutions are preferred to others with respect to some "quality" criterion), and to properly deal with incomplete information (that is, the ability to properly complete missing information with default assumptions, or with assumptions of falsity, or with using some notion of undefinedness). The next example shows the impact of such aspects.

Example 2 (Social Dinner Example). Imagine the organizers of this course planning a fancy dinner for the course participants. To make the event a great success, the organizers decide to ask the attendees to declare their personal wine *preferences*. Soon, the organizers become aware of the fact that there is no wine, which satisfies all of the participant preferences. Thus, they aim at automatically finding the *cheapest* selection of bottles such that any attendee can have her preferred wine at the dinner. This solution should take into account that people usually like wine from their home country, but may not like to drink it abroad.

The organizers quickly realize that several, different specification tools are needed to accomplish this task : in this example, it is more difficult to model the scenario appropriately, and in particular to adequately represent and handle the emerging preferences, priorities, and defaults in absence of complete information, along with conflicts that emerge from them.

This situation motivates a general-purpose approach for modeling and solving also many other problems, which take among others the following aspects into account:

– Possibility of integrating diverse domains;
– Spatial and temporal reasoning (here, the notorious *Frame Problem* is challenging);
– Possibility of modeling constraints;
– Reasoning with incomplete information; and
– Possibility of modeling preferences and priority.

The ASP paradigm has been proposed as a possible solution about ten years ago, as the underlying non-monotonic logic programs are well-positioned to cover these aspects. In the following, we shall briefly look at the roots of ASP and at the relationship of ASP to Prolog, before we turn to the technical preliminaries.

[3] A specification of the Sudoku problem expressed in AnsProlog is reported in Appendix A.

1.1 Roots of ASP

ASP is strongly rooted in the area of Knowledge Representation and Reasoning, and therein in logic programming. However, rather than to foster a general problem solving paradigm, the roots of ASP are in formalisms that aimed at particular representation and reasoning tasks, such as

- modeling an agent's belief sets,
- commonsense reasoning,
- defeasible inferences, and
- preferences and priority.

To this end, many logic-based formalisms for knowledge representation have been developed. As an inherent feature, these formalisms are *nonmonotonic*, that is, they have the property that a growing stock of beliefs may invalidate part of the conclusions that were previously drawn in lack of complete knowledge.

The formalisms, which address above objectives, were motivated by the vision of John McCarthy and other pioneers in AI: logic is an ideal tool for representing and processing knowledge. Oversimplified, the idea can be explained as follows:

- declare knowledge about a "world" of interest by logical sentences;
- more precisely, one should use predicate logic for knowledge representation;
- derive new (implicit) knowledge by an automated inference procedure.

For example, the simple knowledge base

$$K = \{human(socrates), \forall x(human(x) \Rightarrow mortal(x))\}$$

might informally express the fact that Socrates is human and the rules that all humans are mortal in predicate logic; from this knowledge base, we can derive the fact $mortal(socrates)$ using deductive inference procedures, using different methods; *logical calculi* allow us to derive inferences in a purely syntactic way by manipulating formulas according to *inference rules*. In our example, we can infer $mortal(socrates)$ e.g. from the rules of Modus Ponens: $\frac{\phi,\ \phi\Rightarrow\psi}{\psi}$, and Specialisation: $\frac{\forall x(\phi(x)),\ \text{individual } c}{\phi(c)}$.

Loosely speaking, with such a calculus the derivation of new knowledge boils down to simply a search for a proof in terms of inference rule applications from a set of starting axioms. However, a big problem is that, for predicate logic in general, the existence of such a proof is undecidable (as shown in the 1930s by Church) and thus the dream of a "calculus ratiocinator" (or a "thinking machine") in the sense of Leibniz, can not be materialized in general. The insight was that knowledge processing needs control (which inference rule(s) should be applied?) and that often knowledge can be formulated in terms of rules and facts.

1.2 Prolog

After Robinson's breakthrough with the *Resolution principle* in automated theorem proving, in the early 1970s logic programming has been developed as a new knowledge based problem solving paradigm.

Prolog ("Programming in Logic") emerged as a general purpose programming language, whose guiding principle has been popularized by Kowalski's [73] slogan:

ALGORITHM = LOGIC + CONTROL

where the LOGIC on the right hand side stands for the problem specific knowledge, and the CONTROL for the "processing" of that knowledge in a suitable inference procedure.

Computing with Prolog programs is done using a predicate language, featuring the following:

- Terms are used to access objects, where constants stand for individuals (e.g., joe) and variables (e.g., X) for unknown individuals, and function symbols (like in $father(joe)$) are available.
- Terms are used to model basic data structures, like records, e.g $name(joe, doe)$.
- Instead of iteration, there is extensive use of recursion.
- In connection with this, the list constructor $[\cdot|\cdot]$ can be used, which also allows to define higher-order objects (like sets).
- Solutions are obtained via queries (goals) that are posed to the program, where formal proofs provide answers. They build on
  - SLD-resolution, a special variant of the resolution calculus, and
  - unification, as the basic mechanism to manipulate data structures.

The following is a simple Prolog program, familiar from most beginner courses in Prolog, for appending two lists and for reverting a list, respectively.

$$append([\,], X, X). \tag{1}$$
$$append([X|Y], Z, [X|T]) \leftarrow append(Y, Z, T). \tag{2}$$

$$reverse([\,], [\,]). \tag{3}$$
$$reverse([X|Y], Z) \leftarrow append(U, [X], Z), reverse(Y, U). \tag{4}$$

The above program recursively defines the predicates $append(X, Y, Z)$ and $reverse(X, Y)$, where the latter is defined in terms of the former. By posing a query against the program, we then can reverse lists. E.g., to reverse the list $[a, b, c]$, we can pose the query $?\!- reverse([a, b, c], X)$. A proof of the query yields a substitution: $X = [c, b, a]$, which then gives an answer. One can also pose queries that allow to reason backwards from the output to the input (which is not possible in imperative programming). E.g., if we pose $? - reverse([X|a], [b, a])$. the answer substitution $X = b$ tells us that the "input" for the output $[b, a]$ must consist of $[a, b]$.

In principal, above way of programming is a major step forward to our goal of writing programs in a declarative way, but an important point is that it may make a difference how and in which order the clauses of a Prolog programs are given. Although logically equivalent in terms of predicate calculus, if we replace rule (4) above by

$$reverse([X|Y], Z) \leftarrow reverse(Y, U), append(U, [X], Z). \tag{5}$$

and then ask $? - reverse([a|X], [b, c, d, b])$, the evaluation does not terminate (or is stopped because resources are exhausted, with no result). Similar behavior may be found if rules in a program are moved around. This is not a bug of Prolog but intrinsic in its highly efficient inference algorithm (which is sound but incomplete). Operators like the cut (which allow to prune the search space further, at the risk of losing solutions if done improperly), allow the fine control of the evaluation algorithm.

This example raises the legitimate question whether programming in Prolog is truly declarative. In fact, if one keeps in mind the goal of having specifications in which a problem is *declared*, without knowledge on how this declaration will be processed, it is desirable, as far as termination and finding of a solution is concerned, that

– the order of program rules does not matter, and that
– the order of subgoals in a rule body does not matter.

This calls for "pure" declarative programming, in which we (possibly) trade the efficiency of problem solving for strict declarativity of the formalism. The major exponent of this "pure" declarative programming paradigm is the stable model semantics of logic programs, which will be introduced in the sections below.

The stable model semantics is often confused with ASP. Indeed the semantics of the latter has been specified in terms of the former in the seminal paper [64].

The success of ASP is based on the easy usage of ASP as a modeling language, and on the variety of sophisticated algorithms and techniques for evaluating A-Prolog programs, which originated from research on computational complexity of reasoning tasks for such programs. The complexity of ASP reasoning is well understood, and a detailed picture of it and its major extensions can be found in [25]. Advanced AS solvers such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT (see [7]), are able to deal with large problem instances; demonstration efforts of the potential of ASP are made at the AS solver competition [59] which takes place at the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR) since 2007.

1.3 Structure of the Article

The rest of this article is divided into three parts as follows. The first part introduces the stable models semantics of normal logic programs and the answer set semantics of extended logic programs, as well as of extensions thereof. Concepts and notions are given following a historical timeline, which incidentally coincides with the development of increasingly expressive specification languages based on rules. We first recall the least model semantics of Horn logic programming (Section 2) and then turn to the issue of negation in logic programs (Section 3). Then, we consider stratified logic programs, for which the perfect model semantics is the canonical semantics (Section 3.1). We then present the stable model semantics of normal logic programs (Section 4) which coincides with the perfect model semantics on stratified programs (and thus generalizes it). After that, we proceed with some extensions in Section 5; in particular, with constraints, with strong negation—where we arrive at the notion of answer sets—and with disjunctive rule heads.

The second part then considers the ASP paradigm itself. It describes the general idea and shows some ASP programming techniques (Section 6). Furthermore, it overviews AS solvers and their general architecture and implementation principles (Section 7); as an example, we briefly present the AS solver DLV.

The third part is devoted to ASP in the context of the Semantic Web, presenting some formalisms and mentioning some applications in this area (Section 8). The article concludes with issues of current and future ASP research.

2 Horn Logic Programming

We will consider logic programs built from simple constituent blocks, which correspond syntactically to the language of predicate calculus. We will have *constants*, which represent individuals of the domain of discourse, like $sarah$, $chicago$, and 2. They will be represented with lowercase starting letter, or with natural numbers. Variables, like X, $City$, $Name$, denote an individual variable, and are written with uppercase starting letter. Also, one might form *functional terms* combining constants, functions symbols and variables such as in $next(a, Y)$, where $next$ is a binary function symbol.

In some sense, variables and constants can be seen as subjects and objects participating to the scenario we are modeling, which can be tied together through *predicates*, like $hasName$ and $link$. Predicates relate with variables and constants through *atoms*, like $link(chicago, paris)$ or $hasName(C, sarah)$. Note that the former atom has no variables in it (it is *ground*), while the latter is *nonground*. Functional terms are syntactically equivalent to atoms, yet they have different meaning. A (ground) atom is connected to its truth value and acts as a propositional variable: for instance, $hub(rome)$ might be true or false in the sense that $rome$ might be a hub or not; on the other hand, $father(gb)$, when seen as a functional term, denotes an individual of our domain of discourse ("the father of gb"), for which truth or falsity makes no sense in general.

On top of these simple notions we use the idea of *rules*. Rules are grouped in sets that we will call (*logic*) *programs*.

We will start with a class of logic programs featuring the simplest form of a rule.

2.1 Positive Logic Programs

Definition 1 (Positive Logic Program). *A* positive logic program *P is a finite set of clauses (rules) in the form*

$$a \leftarrow b_1, \ldots, b_m \ , \qquad (6)$$

where a, b_1, ..., b_m are atoms of a first-order language L. We call a the head *of the rule, while $b_1, \ldots, b_m$ represents the rule's* body. *A* fact *is a rule with empty body such as $a \leftarrow$, denoted for short as a.*

To give an intuition of the meaning of a rule, a reader familiar with imperative programming languages might interpret this construct as an abstraction of the $\mathbf{if} \ldots \mathbf{then} \ldots$ construct common in traditional programming languages, to which, as it has been illustrated, Modus Ponens might apply. For a reader familiar with first order logic, rules can be seen as material implications restricted to Horn clauses, where $A \leftarrow B$ is read as $B \supset A$ or $B \rightarrow A$.

For instance, the rule

$$connected(cagliari) \leftarrow hub(rome), link(rome, cagliari)$$

might be "procedurally" read as "if Rome is a hub, and there is a link between Rome and Cagliari, then Cagliari is a connected airport," or, when seen as a first-order Horn clause in predicate logic, the same rule can be interpreted as "in any possible scenario in which Rome is a hub and there is a link between Rome and Cagliari, it is the case that Cagliari is connected."

However, we will observe later that rules in declarative logic programming do not strictly correspond to the procedural scheme of imperative languages, nor to material implication. Nevertheless, they are *declarative* constructs, and we make this more clear later in this section.

The above example rule is ground, but logic programs might contain nonground rules like

$$connected(X) \leftarrow hub(Y), link(Y, X) \ ,$$

which can be read as the universally quantified clause $\forall X, Y\ hub(Y) \wedge link(Y, X) \supset connected(X)$. Importantly, one must distinguish between the imperative and logical reading of clauses: a variable X in imperative programming associates a single value to it and stands for a named storage cell, whereas X reads as "any X having a certain property" in the logical interpretation of clauses.

We can also think of a logic program as a description of a scenario, in which certain assertions, either specific and related to certain individuals (that is, ground), or general (that is, nonground, or partially ground), must hold.

The following definitions clarify this intuition.

Definition 2 (Herbrand Universe, Base, Interpretation). *Given a logic program P, the* Herbrand universe *of P, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in P (resp. the vocabulary of L, if explicitly known).*

The Herbrand base *of P, $HB(P)$, is the set of all ground atoms which can be formed from predicates occurring in P and the terms in $HU(P)$. A (Herbrand)* interpretation *is an interpretation I over $HU(P)$, that is, I as subset of $HB(P)$.*

An interpretation can be seen as a set denoting which ground atoms are true in a given scenario.

Example 3. Assume the following program P_1 is given:

$$\begin{aligned} h(0,0). \\ t(a,b,r). \\ p(0,0,b). \\ p(f(X),Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \\ h(f(X),f(Y)) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \end{aligned}$$

The unique function symbol appearing in P_1 is f, and the constant symbols in P_1 are r, a, b, and 0. Thus, $HU(P_1) = \{0, a, b, r, f(0), f(f(0)), \dots, f(a), f(f(a)), \dots\}$, which represents the (infinite) set of individuals possibly involved in P_1.

The Herbrand base is $HB(P_1) = \{p(0,0,0), p(a,a,a), \dots, h(0,0), \dots, t(0,0,0), t(a,a,a), \dots\}$, and represents the set of all possible ground assertions which might hold.

Some possible Herbrand interpretations are

- $I_1 = \emptyset$,
- $I_2 = HB(P_1)$,
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$,

and so on. An interesting question is which scenarios (interpretations) are compatible with P_1. For instance, the interpretation $\{h(0,0), t(a,b,r)\}$ is contradicting P_1, which follows from the simple expectation that, in virtue of the last fact in P_1, also $p(0,0,b)$ should be considered true.

Definition 3. *A* ground instance *of a clause C of the form (6) is any clause C' obtained from C by applying a substitution*

$$\theta\colon \mathit{Var}(C) \rightarrow \mathit{HU}(P)$$

to the variables in C, denoted as $\mathit{Var}(C)$. For any clause C, we denote by $\mathit{grnd}(C)$ the set of all possible ground instances of C, and for any program P we let $\mathit{grnd}(P) = \bigcup_{C \in P} \mathit{grnd}(C)$ (called the grounding of *P).*

Intuitively, $\mathit{grnd}(C)$ allows for the materialization of the universal quantification of variables appearing in C. Roughly speaking, C is a shortcut denoting a set of clauses $\mathit{grnd}(C)$. The range of each variable appearing in C is given by the set of terms appearing in the Herbrand universe.

Example 4. Consider the following program P_2:

$$\begin{aligned} p(f(X),Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).\\ h(0,0).\end{aligned}$$

The ground instances of the first rule in P_2 are

$$\begin{gathered} p(f(0),0,0) \leftarrow p(0,0,0), h(0,0), t(0,0,r).\\ \vdots \\ p(f(0),r,0) \leftarrow p(0,r,0), h(0,r), t(0,0,r).\\ \vdots \\ p(f(r),r,r) \leftarrow p(r,r,r), h(r,r), t(r,r,r).\end{gathered}$$

Definition 4. *Let I be an interpretation. Then I is a* model *of*

- *a ground (variable-free) clause $C = a \leftarrow b_1, \ldots, b_m$, denoted $I \models C$, if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $a \in I$;*
- *a clause C, denoted $I \models C$, if $I \models C'$ for every $C' \in \mathit{grnd}(C)$;*
- *a program P, denoted $I \models P$, if $I \models C$ for every clause $C \in P$.*

Intuitively, a model of P is an interpretation which is compatible with assertions appearing in P.

Example 5. Reconsider the program P_2 in Example 4. Note that $I_1 = \emptyset$ is not a model of P_2 (the fact $h(0,0)$ is not true in I_1), while $I_2 = \mathit{HB}(P_2)$ is a model; indeed, for every program P it clearly holds that $\mathit{HB}(P)$ is a model of P. However, $I_3 = \{h(0,0), t(0,0,r), p(0,0,0)\}$ is not a model of P_2, since the first rule would require $p(f(0),0,0) \in I_3$.

2.2 Minimal Model Semantics

In general, there are multiple "compatible" interpretations of a program P, that is, there can be multiple interpretations, which are models of P. Some of them are however trivial, e.g., think of I_2 in the previous example w.r.t. P_2, or they convey information which is not encoded in P_2. For instance, $I_4 = I_3 \cup \{p(f(0), 0, 0), h(r, r)\}$ is a model of P_2. There is however no evidence that $h(r, r)$ should be true according to P_2: indeed we might remove it from I_4, obtaining a smaller model $I_5 = I_3 \cup \{p(f(0), 0, 0)\}$.

On the other hand, we cannot remove $p(f(0), 0, 0)$ from I_5 since the first rule of the program would not be satisfied. In other words, $p(f(0), 0, 0)$ is an atom which has to be *necessarily* true in the scenario described by P_2, while this is not the case for $h(r, r)$.

One might ask at this point whether there exists a particular *canonical model* for a program which contains only the atoms which are necessarily true according to P. This notion of "necessity" is commonly called *foundedness*.

Example 6. Consider the small program P_3

$$a \leftarrow b. \quad b \leftarrow c. \quad c.$$

The truth of atom a in the model $I = \{a, b, c\}$ is "founded." Intuitively, c must appear in any model of P_3, which implies that also b and then a are necessarily true.

Given the program P_4

$$a \leftarrow b. \quad b \leftarrow a. \quad c.$$

we obtain that the truth of atom a in model $I = \{a, b, c\}$ is not founded. In other words, there is no necessity of a appearing in a model. Indeed, $I' = \{c\}$ is also a model.

The above intuition can be translated into a formal semantics, which prefers models having as few true facts as is possible.

Definition 5. *A model I of a program P is* minimal*, if there exists no model J of P such that $J \subset I$.*

Theorem 1. *Every positive logic program P has a single minimal model (called the* least model*), denoted $LM(P)$.*

This is entailed by the following property:

Proposition 1. *If I and J are models of P, then also $I \cap J$ is a model of P.*

Example 7. For $P_3 = \{a \leftarrow b. \quad b \leftarrow c. \quad c.\}$, we have $LM(P_3) = \{a, b, c\}$. For $P_4 = \{a \leftarrow b. \; b \leftarrow a. \; c.\}$, we get the least model $LM(P_4) = \{c\}$

For program P_1 above, we have

$$LM(P_1) = \{h(0, 0), t(a, b, r), p(0, 0, b), p(f(0), 0, a), h(f(0), f(0))\} \ .$$

Computation of the Least Model. A natural question is, how we can compute the least model $LM(P)$ of a program P.

By means of the *immediate consequence operator*, one can obtain $LM(P)$ through an iterative process. Let $T_P\colon 2^{HB(P)} \to 2^{HB(P)}$ be an operator defined as

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there exists some } a \leftarrow b_1, \ldots, b_m \\ \text{in } grnd(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \end{array} \right\} .$$

We define $T_P^0 = \emptyset$, and $T_P^{i+1} = T_P(T_P^i)$ for $i \geq 0$.

Theorem 2. *T_P has a least fixpoint, $lfp(T_P)$, and the sequence $\langle T_P^i \rangle$, $i \geq 0$, converges to it, i.e., $lfp(T_P) = LM(P)$.*

The above result can be proved by means of the fixpoint theorems of Knaster-Tarski and of Kleene given in Appendix B. The second part of the theorem is easily shown by observing that $lfp(T_P)$ is a model of P and no smaller model exists.

Example 8. The immediate consequence operator captures the idea that if all the atoms in a rule r body are founded, then also the head of r must be founded.

For instance, for $P_3 = \{a \leftarrow b. \quad b \leftarrow c. \quad c.\}$, we have

$$T_{P_3}^0 = \{\}, \quad T_{P_3}^1 = \{c\}, \quad T_{P_3}^2 = \{c, b\}, \quad T_{P_3}^3 = \{c, b, a\}, \quad T_{P_3}^4 = T_{P_3}^3 \ .$$

Hence, $lfp(T_{P_3}) = \{c, b, a\}$. For $P_4 = \{a \leftarrow b. \quad b \leftarrow a. \quad c.\}$, we have

$$T_{P_4}^0 = \{\}, \quad T_{P_4}^1 = \{c\}, \quad T_{P_4}^2 = T_{P_4}^1 \ .$$

Hence $lfp(T_{P_4}) = \{c\}$.

For program P_1 above, we have

$$\begin{aligned} T_{P_1}^0 &= \emptyset, \\ T_{P_1}^1 &= \{h(0,0), t(a,b,r), p(0,0,b)\} \\ T_{P_1}^2 &= \{h(0,0), t(a,b,r), p(0,0,b), p(f(0),0,a), h(f(0), f(0))\} \\ T_{P_1}^3 &= T_{P_1}^2 . \end{aligned}$$

3 Negation in Logic Programs

Positive logic programs allow for declarative modeling of a variety of problems. However, it turns out that many situations require a construct which model the intuitive notion of negation. Negation is a natural linguistic concept and happens to be extensively required when natural problems have to be modeled declaratively. For instance, given the rule

$$connected(X) \leftarrow hub(Y), link(Y, X) \ ,$$

which defines airports connected to at least one hub airport, one might think of defining airports which are *not* connected to any hub. This can be modeled intuitively by put the not modifier in front of atoms, and considering the rule

$$badlyConnected(X) \leftarrow \mathrm{not}\ connected(X) \ .$$

We will define *normal logic programs* as a set of clauses having the form

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n \qquad (n, m \geq 0) \tag{7}$$

where a and all b_i, c_j are atoms in a first-order language L. Note that rule bodies now include expressions which we call *(default) negated literals* $\text{not } c_1, \dots, \text{not } c_l$, which consist of atoms c_i preceded by the negation modifier not. Accordingly, the atoms $b_1, \dots, b_k$ are called *positive literals*.

Intuitively, a ground literal corresponds to a propositional variable as it was the case for atoms: a negated literal has a truth value which is opposite to its corresponding positive literal. For instance, if $hub(rome)$ is true, then $\text{not } hub(rome)$ is false.

Once negated literals are syntactically defined, one can think of a proper formal meaning for rules in which they appear. The Prolog semantics has been pragmatically and operationally extended from SLD to SLDNF in terms of *Negation as failure*: here, one considers as false a negated literal $\text{not } a(\cdot)$, if the truth of its corresponding positive literal cannot be (finitely) proved through SLD resolution.

It is important to observe that negation in classical logic is different from negation in logic programming (cf. surveys [14,5] and [44,41] for more discussion).

Example 9. Consider the program P_5:

$$\begin{aligned} man(dilbert).& \\ single(X) \leftarrow man(X), \text{not } husband(X).& \\ husband(X) \leftarrow fail.& \quad \text{\% fail = ''false'' in Prolog} \end{aligned}$$

Under Prolog semantics, if we ask the query

$$?- single(X).$$

we obtain as an answer

$$X = dilbert \ .$$

Intuitively, the answer is motivated by the fact that $husband(dilbert)$ cannot be proved from P_5. For proving $single(dilbert)$ using forward chaining, one can use the first rule of the program, in which it must be first shown that $man(dilbert)$ holds, and then that there is no proof for $husband(dilbert)$; indeed, there is no evidence that $husband(dilbert)$ is true in P_5.

Note however that this operational approach fails to give a satisfactory answer for programs like P_6

$$\begin{aligned} man(dilbert).& \\ single(X) \leftarrow man(X), \text{not } husband(X).& \\ husband(X) \leftarrow man(X), \text{not } single(X).& \end{aligned}$$

where $single(dilbert)$ and $husband(dilbert)$ are mutually dependent using negation. An SLD resolution algorithm would loop forever when trying to answer the query $single(X)$.

Approaches which give meaning to logic programs via a model theoretic definition (that is, providing an appropriate notion for a "best" model) are able to treat recursive definitions for positive programs properly, for which a unique minimal model exists. However, P_6 has two minimal Herbrand models

$$\begin{aligned} M_1 &= \{man(dilbert), single(dilbert)\}, \text{ and} \\ M_2 &= \{man(dilbert), husband(dilbert)\}\ . \end{aligned}$$

Both M_1 and M_2 satisfy P_6 and constitute a minimal set of necessarily true facts which are compatible with P_6. One thus may guess that introducing negation in logic programs induces a major problem regarding the meaning of normal logic programs.

The debate about the proper semantics for attributing meaning to negation in logic programs has been long lasting,[4] and provoked what we could call the *Great Logic Programming Schism*. Indeed, there are two philosophically very different approaches:

1. To keep the idea of defining a single model for a program, possibly including also problematic classes of programs with negation. This can be achieved by properly defining which *single* model should be selected among all classical models of a program. This line of research produced the notion of *perfect model* [109] which has been agreed being satisfactory for the class of so-called "stratified programs." For general normal problems, the most popular semantics is perhaps the one based on the *well-founded model* [122].
2. To identify a collection of *multiple* preferred models. This line of research abandons the "dogmatic" requirement of a single model and accepts the possibility of having multiple scenarios compatible with a given program. Note that, in general, for such multiple models approaches, we have a single model for positive and stratified programs which corresponds to the least and perfect model, respectively.

Answer Set Programming and its underlying *stable model semantics* is based on the latter methodology.

3.1 Stratified Negation

As a first class of programs with negation we will consider *stratified programs* [4]. Stratified programs have the property that one can find an ordering for the evaluation of the rules in the program, such that the value of negative literals can be predetermined.

Intuitively, for evaluating the body of a rule containing $\text{not}\, r(\boldsymbol{t})$, the value of the negative literal $r(\boldsymbol{t})$ should be known. This mimics the negation-as-failure approach as follows:

1. First evaluate $r(\boldsymbol{t})$;
2. if $r(\boldsymbol{t})$ is false, then $\text{not}\ r(\boldsymbol{t})$ is true;
3. if $r(\boldsymbol{t})$ is true, then $\text{not}\ r(\boldsymbol{t})$ is false and the rule is not applicable.

Example 10. We can evaluate the single rule program

$$boring(chess) \leftarrow \text{not}\ interesting(chess)$$

[4] The interested reader might refer to [14,31,32] for surveys about the matter.

$$\begin{tikzcd}
husband \arrow[r] \arrow[dr] & married \\
single \arrow[u, "\star"] \arrow[r] & man
\end{tikzcd}$$

Fig. 2. $dep(P_7)$

according to this recipe: as $interesting(chess)$ clearly evaluates to false, the negated literal $\text{not}\ interesting(chess)$ evaluates to true; hence, also $boring(chess)$ evaluates to true. This results in the Herbrand model $H = \{boring(chess)\}$ of P, which is the intuitive meaning of P.

Note however that this implicitly introduces a particular order of evaluation for rules and make specifications *procedural* more than declarative.

Dependency Graph. The above method makes only sense if there is no cyclic negation in programs. Otherwise, it is not possible to find an "evaluation ordering" for a program. The notion of dependency graph of programs captures this intuition.

Definition 6 (Dependency graph). *The dependency graph of a program P, $dep(P) = \langle V, E\rangle$, consists of*

- *a set of nodes V, which is defined as the set of all predicates p occurring in P, and*
- *a set of arcs E, which contains arcs of form $p \rightarrow q$ if and only if an atom with predicate name p is in the head of a rule $r \in P$ and the body of r contains a literal with predicate name q. If this literal is under negation, the edge will be marked with $\star$ ($p \rightarrow^\star q$).*

Example 11. Consider the following program P_7

$$\begin{aligned} man(dilbert).& \\ husband(X) \leftarrow man(X),\ married(X).& \\ single(X) \leftarrow man(X), \text{not}\, husband(X).& \end{aligned}$$

and its dependency graph $dep(P_7)$ shown in Figure 2. The order of evaluation for negated predicates is built according to the following policy: If there is a path in $dep(P_7)$ from a predicate $p = p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{n-1} \rightarrow p_n$ to a predicate $q = p_n$, such that some $p_i \rightarrow p_{i+1}$ is marked with $\star$, then q must be evaluated prior to p. In this example we have a path $single \rightarrow^\star husband \rightarrow married$, thus both $husband$ and $married$ must be evaluated before $single$.

Stratification. We formalize the notion of stratification as follows. Let $pred(R)$ denote the set of predicate names occurring in a set of rules R.

Definition 7 (Stratification). *A* stratification *of a set of rules P is a partitioning $\Sigma = \{S_i \mid i \in \{1, \ldots, n\}\}$ of $pred(P)$ into n nonempty and pairwise disjoint sets of predicate names such that*

(a) if $p \in S_i$, $q \in S_j$, and $p \rightarrow q$ is in $dep(P)$ then $i \geq j$; and
(b) if $p \in S_i$, $q \in S_j$, and $p \rightarrow^\star q$ is in $dep(P)$ then $i > j$.

The sets $S_1, \dots, S_n$ are called the strata *of P w.r.t. Σ. A program P is called* stratified, *if it has some stratification Σ.*

Note that there are programs which are not stratified, such as P_6 above. The stratification Σ specifies an *evaluation order* for the predicates in a logic program. Here *evaluation* of a predicate p means to compute the set of true atoms that have p as predicate name. This sequential evaluation can be done by computing a series of *iterative least models*.

Definition 8. *Let P a logic program with a stratification $\Sigma = \{S_1, \dots, S_k\}$ of length $k \geq 1$. We define P_{S_i} as the subset of the rules of P which have a head atom whose predicate belongs to S_i, and $HB^\star(P_{S_i}) = \bigcup_{j \leq i} \{p(\mathbf{t}) \in HB(P) \mid p \in S_j\}$. We define the* iterative least models *$M_i \subseteq HB(P)$ with $i \in \{1, \dots, k\}$ by:*

(i) M_1 is the least model of P_{S_1};
(ii) if $i > 1$, then M_i is the least subset M of $HB(P)$ such that (a) M is a model of P_{S_i}, and (b) $M \cap HB^\star(P_{S_{i-1}}) = M_{i-1} \cap HB^\star(P_{S_{i-1}})$.

We denote by $M_{P,\Sigma}$ the iterative least model M_k.

Example 12. Consider again the program P_7:

$$\begin{aligned} &man(dilbert). \\ husband(X) &\leftarrow man(X),\ married(X). \\ single(X) &\leftarrow man(X),\ \text{not}\ husband(X). \end{aligned}$$

According to the dependency graph $dep(P_7)$, a stratification Σ for P_7 is

$$S_1 = \{man, married\},\ S_2 = \{husband\},\ S_3 = \{single\}\ .$$

We obtain $M_1 = LM(P_{S_1}) = \{man(dilbert)\}$ from the evaluation of $P_{S_1} = \{man(dilbert)\}$. When evaluating M_2 we obtain

$$P_{S_2} = \{husband(X) \leftarrow man(X), married(X)\}\ .$$

Note that $HB^\star(P_{S_1}) = \{man(dilbert), married(dilbert)\}$. It is easy to see that $M_2 = \{man(dilbert)\}$ is a model for P_{S_2}, and that $M_2 \cap HB^\star(P_{S_1}) = M_1 \cap HB^\star(P_{S_1})$; also, M_2 is the least model having these properties.

For the evaluation of M_3, note that

$$P_{S_3} = \{single(X) \leftarrow man(X),\ \text{not}\ husband(X)\}\ .$$

Thus one finds that $M_3 = \{single(dilbert)\} \cup M_2$ is the least model of P_{S_3} such that $M_3 \cap HB^\star(P_{S_2}) = M_2 \cap HB^\star(P_{S_2})$.

It is worth noting that stratifications are not unique. For instance, one can compute the iterative least models using an alternative stratification Σ', in which $S_1 = \{man, married, husband\}$ and $S_2 = \{single\}$.

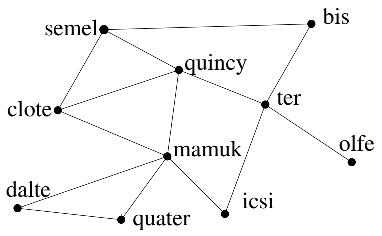


Fig. 3. An example railroad network

In both cases the iterative least model obtained at the last iteration is the same. An important result tells us that, provided a stratification exists, other stratifications produce the same final model.

Theorem 3 ([4]). *Let P be a stratified program. Then for every stratifications Σ and Σ' of P, it holds that $M_{P,\Sigma} = M_{P,\Sigma'}$.*

Hence, we can drop the dependency of $M_{P,\Sigma}$ on a given stratification Σ and define $M_P = M_{P,\Sigma}$ (for a Σ of choice) as the canonical model for P, which is referred to as *perfect model* [109].[5]

Example 13 (Railroad network). Take, as an example, the railroad network given in Figure 3. The goal is to determine whether safe connections between locations are possible. Given two railroad stations a and b, a *cutpoint station* c for a and b is such that if connections to c fail, there is no alternative connection between a and b. We will say that the connection between a and b is safe if there are no cutpoints between a and b. In Figure 3, ter is a cutpoint for $olfe$ and $semel$, while $quincy$ is not.

The above problem can be modeled as follows. First, we introduce the set of predicates:

- $station(a)$: a is a railway station;
- $link(a, b)$: there is a direct connection from station a to b;
- $linked(a, b)$: the symmetric closure of $link$; that is, $linked(a, b)$ and $linked(b, a)$ hold whenever $link(a, b)$ holds;
- $connected(a, b)$: there is path linking a to b, either direct or through intermediate stations;
- $cutpoint(x, a, b)$: each existing path from a to b goes through station x;
- $circumvent(x, a, b)$: when going from a to b one can avoid x; that is, there is a path between a and b not passing from x;
- $has\_icut\_point(a, b)$: there is at least one cutpoint between a and b;
- $safely\_connected(a, b)$: a and b are connected with no cutpoint.

We will assume that atoms of form $link(a, b)$ are given as set of facts describing the railroad network at hand. Other predicates are defined according to the program P_r

[5] In fact, Przymusinski and Apt et al. developed their semantics independently, but the proposals coincide on stratified programs, and the name *perfect model* for M_P is customary.

$$
\begin{aligned}
linked(A,B) &\leftarrow link(A,B). && (R_1)\\
linked(A,B) &\leftarrow link(B,A). && (R_2)\\
connected(A,B) &\leftarrow linked(A,B). && (R_3)\\
connected(A,B) &\leftarrow connected(A,C), linked(C,B). && (R_4)\\
cutpoint(X,A,B) &\leftarrow connected(A,B), station(X), && (R_5)\\
&\qquad \text{not } circumvent(X,A,B). &&\\
circumvent(X,A,B) &\leftarrow linked(A,B), X \neq A, station(X), X \neq B. && (R_6)\\
circumvent(X,A,B) &\leftarrow circumvent(X,A,C), circumvent(X,C,B). && (R_7)\\
has\_icut\_point(A,B) &\leftarrow cutpoint(X,A,B), X \neq A, X \neq B. && (R_8)\\
safely\_connected(A,B) &\leftarrow connected(A,B), && (R_9)\\
&\qquad \text{not } has\_icut\_point(A,B). &&\\
station(X) &\leftarrow linked(X,Y). && (R_{10})
\end{aligned}
$$

Fig. 4. Railroad program P_r

shown in Figure 4.[6] Informally, R_1 and R_2 define *linked* as the symmetric closure of *link*, and *connected* is defined by means of rules R_3 and R_4. Roughly speaking, R_3 expresses that a and b are *connected* if there is a direct *link* among them, while R_4 expresses that a and b are *connected* if there is a node c, which a is connected to, and c has a link to b. Negation is exploited in R_5 for defining *cutpoint*s: x is a cutpoint for all the paths from a to b if a and b are *connected* and x is a *station* for which $circumvent(x, a, b)$ does not hold.

Now, let us analyze how to define the notion of *circumvention*. There are two ways for circumventing a station x when going from a to b: either there exists a direct *link* from a to b (rule R_6) or one can *circumvent* x when going from a to c and then *circumvent* x when going from c to b (rule R_7).

Accordingly, the path from a to b *has a cutpoint* if there is a nontrivial (i.e., x is neither equal to a or b) cutpoint from a to b (rule R_9). Again, negation is exploited for defining when a and b are safely connected (rule R_9): couples of endpoint stations are safely connected if they are *connected* and do not have cutpoints. Eventually, rule R_{10} defines a *station* as those nodes which are directly *linked* to others.

The dependency graph of P_r is shown in Figure 5. A possible stratification of P_r is $\Sigma_r = \{S_1, S_2, S_3\}$, where

- $S_1 = \{link, linked, station, circumvent, connected\}$,
- $S_2 = \{cutpoint, has\_icut\_point\}$, and
- $S_3 = \{safely\_connected\}$.

We then get the iterative least models

- $M_1 = \{\, linked(semel, bis), linked(bis, ter), linked(ter, olfe), \ldots,$
  $connected(semel, olfe), \ldots, circumvent(quincy, semel, bis), \ldots \}$,

[6] The predicate $\neq$ is a "built-in" predicate, which cannot be user defined. It is thus not shown in the evaluation and the dependency graph.

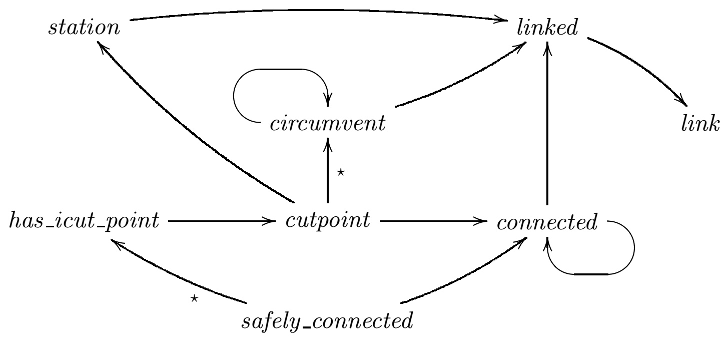


Fig. 5. Dependency graph $dep(P_r)$ of the railroad program P_r

- $M_2 = M_1 \cup \{\, cutpoint(ter, semel, olfe), has\_icut\_point(semel, olfe), \ldots \}$, and
- $M_3 = M_2 \cup \{\, safely\_connected(semel, bis), safely\_connected(semel, ter) \,\}$.

The iterative least model M_3 is then a perfect model for P_r. Note that M_3 does not contain $safely\_connected(semel, olfe)$.

3.2 Unstratified Negation

The notion of perfect model is however inadequate whenever a program has no stratification. This happens when two or more predicates are mutually defined over "not," like in the following program P_u:

$$\begin{aligned} & man(dilbert). \\ single(X) &\leftarrow man(X), \text{not } husband(X). \\ husband(X) &\leftarrow man(X), \text{not } single(X). \end{aligned}$$

Note that P_u has two minimal models (which, as shown next, are *stable*):

- $M = \{man(dilbert),\ single(dilbert)\}$ and
- $N = \{man(dilbert),\ husband(dilbert)\}$;

both might be seen as "plausible" scenarios compatible with P_u.

In general, we can associate to a program P a set of *preferred* (or plausible) models $PM(P)$. In the presence of multiple plausible models, each describing a possible scenario specified by a given program, a natural question is how to interpret and how to reconcile possible discrepancies between models appearing in $PM(P)$.

One can consider this issue from two complementary points of view:

1. One point is to see P as a knowledge base, in which explicit (facts) and implicit (rules) information is stored, and wonder if a given query q (or, in general, a formula) holds. Queries can be ground (e.g., $q = man(dilbert)$ holds if q is true w.r.t. P_u according to some criterion), or nonground (e.g., for evaluating $q = man(X)$ we have to find the set of values x such that $man(x)$ holds in P_u).

In this respect, a ground query q can be answered under *Cautious (Skeptical) Reasoning*, that is q evaluates to true if it is true in *every* model in $PM(P)$, or under *Brave (Credulous) Reasoning*, in which q is true if it is true in *some* preferred model. Similarly, answering a non-ground query q amounts to finding the set of all the ground assignments of q which hold in any preferred model (cautious reasoning) or in some preferred model (brave reasoning).

2. Cautious and brave reasoning can be seen as a form of quantification/iteration over preferred models, which however still depict a single scenario. In cautious reasoning the single scenario (the set of true facts) is described by the intersection of all the models, while in brave reasoning one considers their union, this way discarding the richer information given in $PM(P)$.

   However, each model in $PM(P)$ brings peculiar information: it can be seen as the representation of a possible world compatible with P, or, in other words, as a *solution* to the problem instance encoded by P. *Model generation* (that is, the computation of the set $PM(P)$) in this respect is—more than query answering—of valuable importance.

Example 14. The preferred models M and N of P_u represent "possible worlds" compatible with P_u. The ground atom $man(dilbert)$ is a cautious and brave consequence of P_u. But, neither $single(dilbert)$ nor $husband(dilbert)$ are cautious consequences, whereas both are brave consequences of P_u (the first holds in M while the second holds in N).

4 Stable Semantics

Many definitions for $PM(P)$ have been conceived in the past, cf. [14,94]. We will concentrate from this point on the—largely considered the most prominent one—notion of preferred model based on *stable models*.

4.1 Normal Logic Programs – Syntax

A logic program P based on the stable model semantics has the same syntactic building blocks as stratified programs: importantly, it is not necessary that P has a stratification, as we do not rely on the notion of perfect model for computing its semantics. Also, we keep the the notions of Herbrand universe $HU(P)$, Herbrand base $HB(P)$, and interpretation as for not-free ("positive") logic programs.

4.2 Stable Model Semantics

First, we will define the stable model semantics for a variable-free (ground) program.

The intuition behind stable model semantics is to treat negated atoms in a special way. Intuitively, such atoms are a source of "contradiction" or "unstability."

Example 15. In P_u from above, one can consider $M' = \{man(dilbert)\}$ as possible, preferred model. Assuming facts in M' as true, note however that the two rules of P_u would enforce to assume that besides $man(dilbert)$ also $single(dilbert)$ and

$husband(dilbert)$ are true. On the other hand, if one considers $M'' = \{man(dilbert), single(dilbert), husband(dilbert)\}$ as the set of true facts, it turns out that the two rules of P_u have now their bodies false, and do not give evidence of truth for $single(dilbert)$ and $husband(dilbert)$.

"Stability" can thus be seen as follows: if an interpretation M of P is not—in the sense formalized below—self-contradicting, then it is *stable*.

Definition 9. *The* Gelfond-Lifschitz reduct *[63] (short GL-reduct or simply reduct) of a program P w.r.t. an interpretation M, denoted P^M, is a program obtained by*

1. *removing rules with* $\text{not}\ a$ *in the body for each $a \in M$; and*
2. *removing literals* $\text{not}\ a$ *from all other rules.*

Intuitively, given an interpretation M, the conditions 1 and 2 above enforce truth values for negative literals. If $a \in M$, then a rule's body with the negative literal $\text{not}\ a$ cannot become true. On the other hand, if $a \notin M$, then $\text{not}\ a$ can be assumed true and removed from any body where it occurs.

In other words, M can be seen as an *assumption* about which negated literals are true and what are false; the program P^M incorporates these assumptions. Note that P^M is a positive program, and thus has a least model $LM(P^M)$. If P^M does not "contradict" M, one should expect that $LM(P^M) = M$, that is, M can be reconstructed from scratch applying the rules of P^M. If this happens to be the case, then M can be regarded as being "stable."

Definition 10. *An interpretation M of P is a* stable model *of P, if*

$$M = LM(P^M).$$

Note that $P^M = P$ for any "not"-free program P. Thus, $LM(P)$ (which is equal to $LM(P^M)$) is its single stable model.

Example 16. If we take P_u again in consideration

$$man(dilbert). \qquad (f_1)$$
$$single(dilbert) \leftarrow man(dilbert), \text{not}\ husband(dilbert). \qquad (r_1)$$
$$husband(dilbert) \leftarrow man(dilbert), \text{not}\ single(dilbert). \qquad (r_2)$$

we may have the following "candidate" interpretations:

- $M_1 = \{man(dilbert), single(dilbert)\}$,
- $M_2 = \{man(dilbert), husband(dilbert)\}$,
- $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$
- $M_4 = \{man(dilbert)\}$,

One can verify that only M_1 and M_2 qualify themselves as stable models.

- if we consider M_1 we get that the reduct $P_u^{M_1}$ is

$$man(dilbert).$$
$$single(dilbert) \leftarrow man(dilbert).$$

Note that $husband(dilbert) \notin M_1$, thus not $husband(dilbert)$ is removed from r_1. On the other hand r_2 is deleted from P_u since $single(dilbert) \in M_1$: indeed, under the assumption made in M_1, the literal not $husband(dilbert$ is false and will prevent r_2 to trigger and make its head true.

The least model of $P_u^{M_1}$ is $\{man(dilbert), single(dilbert)\}$ which coincides with M_1.

Symmetrically, we can verify that M_2 is stable as well.

– On the other hand, M_3 and M_4 are not stable. If we take $M_3 = \{man(dilbert),$ $single(dilbert)$, $husband(dilbert)\}$ in consideration, we find that $P_u^{M_3}$ consists only of $man(dilbert)$. Both r_1 and r_2 are indeed deleted. Thus, $LM(P_u^{M_3}) = \{man(dilbert)\} \neq M_3$. This means that the assumptions made in M_3 are not "stable" with respect to negated literals in P_u.

If we take $M_4 = \{man(dilbert)\}$, we observe that $P_u^{M_4}$ consists of

$$\begin{aligned} &man(dilbert).\\ single(dilbert) \leftarrow\ &man(dilbert).\\ husband(dilbert) \leftarrow\ &man(dilbert). \end{aligned}$$

given that both not $husband(dilbert)$ and not $single(dilbert)$ are removed from r_1 and r_2 respectively. Therefore, $LM(P_u^{M_4}) = \{man(dilbert), single(dilbert),$ $husband(dilbert)\} \neq M_4$.

Notably, there are situations in which "stability" is impossible and no meaning can be assigned to a program.

Example 17. The program P_i

$$p \leftarrow \text{not } p. \tag{8}$$

has no stable models. Consider any interpretation M for P_i such that $p \notin M$. Thus, not p is true and the body of (8) is satisfied, which means that p should be true as well in order for M being a model for P_i. But this is in direct contradiction to $p \notin M$. Now, if we take an interpretation M' such that $p \in M'$, we get that not p is false and our rule (8) is satisfied, hence M' is a model for P_i. But it is not a stable model, as the reduct $P_i^{M'} = \emptyset$, and we have that $LM(P_i^{M'}) = \emptyset$, which is different from M'.

If we take an arbitrary program P, and add the rule (8) (with p being a new propositional atom), we get that P has no stable model.

Example 18. Consider the program P_s:

$$s \leftarrow \text{not } q. \tag{r_1}$$
$$q \leftarrow \text{not } s. \tag{r_2}$$
$$p \leftarrow q, \text{not } s. \tag{r_3}$$
$$f \leftarrow s, \text{not } f. \tag{r_4}$$

P_s has a single stable model $M_1 = \{p, q\}$, while $M_2 = \{s\}$ is not stable.

- Indeed, for $M_1 = \{p, q\}$ we have that in $P_s^{M_1}$ the rules r_1 and r_4 are deleted, while r_2 and r_3 are modified, obtaining:

$$q.$$
$$p \leftarrow q.$$

For which $LM(P_i^{M_1}) = \{p, q\} = M_1$.
- For $M_2 = \{s\}$, we get $P_s^{M_2}$ by deleting r_2 and r_3 from P_s and updating r_1 and r_4:

$$s.$$
$$f \leftarrow s.$$

We get $LM(P_s^{M_2}) = \{s, f\} \neq M_2$. Note that $M_3 = \{s, f\}$ is not stable as well. Indeed, one can observe that rule r_4 prevents the existence of a stable model containing s.

Programs with Variables. As for the case of positive and stratified programs, it is immediate to lift the notion of stable model from propositional programs to non-ground ones. Intuitively, this step amounts to considering non-ground rules (containing variables) as shorthands for all their possible ground instances, obtained using a domain of choice for the terms which can be constructed. This latter domain is usually the Herbrand universe of the program at hand. The stable semantics of non-ground programs is thus obtained by means of a reduction to the variable-free case.

Definition 11. *Given a program P, an interpretation M of P is a* stable model *of P, if M is a stable model of $grnd(P)$.*

Example 19. Consider the following variant of P_u which we will call $P_{u'}$:

$$man(dilbert). \qquad (r_1)$$
$$woman(alice). \qquad (r_2)$$
$$single(X) \leftarrow man(X), \text{not } husband(X). \qquad (r_3)$$
$$husband(X) \leftarrow man(X), \text{not } single(X). \qquad (r_4)$$

We have that, for instance,

$$\begin{aligned} grnd(r_3) = \{\ & single(dilbert) \leftarrow man(dilbert), \text{not } husband(dilbert).\\ & single(alice) \leftarrow man(alice), \text{not } husband(alice). \qquad \}; \end{aligned}$$

$$\begin{aligned} grnd(P_{u'}) = \{\ & man(dilbert).\\ & woman(alice).\\ & single(dilbert) \leftarrow man(dilbert), \text{not } husband(dilbert).\\ & single(alice) \leftarrow man(alice), \text{not } husband(alice).\\ & husband(dilbert) \leftarrow man(dilbert), \text{not } single(dilbert).\\ & husband(alice) \leftarrow man(alice), \text{not } single(alice). \qquad \}. \end{aligned}$$

The program $grnd(P_{u'})$, and thus $P_{u'}$, has the following stable models:

- $M_1 = \{man(dilbert), woman(alice), single(dilbert)\}$
- $M_2 = \{man(dilbert), woman(alice), husband(dilbert)\}$

4.3 Semantic Properties of Stable Models

The success of stable models as semantics for normal logic programs (with arbitrary usage of negation) relies on two important aspects: first, stable models have a strong theoretical basis, and enjoy many properties which reflect natural intuitions. Second, as it will be seen in Section 6 they pave the way to a innovative problem modeling methodology.

We survey here some important (most of which desirable) theoretical properties of stable models. The reader can refer to [83,55,61] for other insights, alternative definitions and properties of stable models.

We first consider the relationship between stable models and classical models of a logic program, i.e., when negation as failure is interpreted as classical negation.

To this end, the notion of (classical) Herbrand model is easily lifted to clauses with negated literals in their bodies.

Definition 12. *Let I be an interpretation. Then I is a* model *of*

- *a ground clause $C : a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$, denoted $I \models C$, if either $\{b_1, \ldots, b_m\} \nsubseteq I$ or $\{a, c_1, \ldots, c_n\} \cap I \neq \emptyset$.*
- *a clause C, denoted $I \models C$, if $I \models C'$ for every $C' \in grnd(C)$;*
- *a program P, denoted $I \models P$, if $I \models C$ for every clause C in P.*

Intuitively, the above definition lifts Definition 4 by taking in consideration negated literals: an interpretation I is, again, "compatible" with a clause C either if it contains the head of C, or if the body of C is false. A body can be false either if some positive b_i is not in I, or if some c_i is in I. One expects that if the body of C is true, then also its head must be true: indeed, if $b_1, \ldots, b_m \in I$ and $c_1, \ldots, c_n \notin I$, I can be model of C only if it contains a.

The above definition complies with the notion of Herbrand model satisfying the clause $a \vee b_1 \vee \ldots \vee b_m \vee \text{not } c_1 \vee \ldots \vee \text{not } c_n$, where not is interpreted as classical negation. Now the following property holds:

Theorem 4

1. *Every stable model M of P is a model of P.*
2. *A stable model M does not contain any model M' of P properly ($M' \not\subset M$), i.e., is a minimal model of P (w.r.t. $\subseteq$).*

The above properties guarantee that stable models of a program with negation enjoy two of the desirable properties holding for least models of positive programs: first, a stable model M of P is "compatible" with all the rules of P, that is, it does not contradict P. Also, M contains a minimal amount of facts which one must admit to be true for gaining the "compatibility" with the scenario described by P, and no unnecessary and/or redundant information.

Corollary 1. *Stable models are incomparable w.r.t. $\subseteq$, i.e., if M_1 and M_2 are different stable models of P, then $M_1 \nsubseteq M_2$ and $M_2 \nsubseteq M_1$.*

Also, stable models gracefully generalize the semantics for positive programs (the least model of a positive program P is clearly the unique stable model of P), and for stratified semantics: indeed, the perfect model of a stratified program is also its unique stable model.

Theorem 5. *If a program P is stratified, then P has a single stable model, which coincides with the perfect model.*

Note, for instance, that the railroad program P_r is stratified. Its single stable model coincides with the perfect model. It is indeed worth noting that there is only one stable configuration for a stratified program although it can have multiple minimal models.

Example 20. If one considers the program P_m

$$p(a).$$
$$r(X) \leftarrow p(X), \text{not}\, q(X).$$

we get two minimal models $M_1 = \{p(a), r(a)\}$ and $M_2 = \{p(a), q(a)\}$ for P_m. Note that while M_1 is stable, M_2 is not stable, as the reduct $grnd(P_m)^{M_2} = \{p(a)\}$, and $LM(grnd(P_m)^{M_2}) = \{p(a)\} \neq M_2$.

What makes M_2 different from M_1 is the fact that there is neither rule nor fact in P_m justifying the presence of $q(a)$ in a model.

Indeed one can see stable models as models in which all atoms $a \in M$ are somehow "supported" by evidence: in a sense, a stable model "supports", or "gives evidence" of the truth of each $a \in M$.

Theorem 6. *Given a program P and an interpretation I, let*

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \textit{there is some } r = a \leftarrow b_1, \ldots, b_m, c_1, \ldots, \text{not}\, c_n \in grnd(P) \\ \textit{such that } \{b_1, \ldots, b_m\} \subseteq I, \{c_1, \ldots c_m\} \cap I = \emptyset \end{array} \right\} .$$

If I is a stable model of P, then $T_P(I) = I$.

Example 21. Note that $q(a)$ in example 20 is *unsupported* in M_2, indeed $q(a) \notin T_P(M_2)$.

Nonetheless, it must be noted that there are models which are minimal fixed points of T_P, but are however not stable:

Example 22. Consider the short program P_s:

$$a \leftarrow \text{not}\, b.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

Note that $M_1 = \{a\}$ and $M_2 = \{b, c\}$ are both minimal and such that $T_{P_s}(M_1) = M_1$ and $T_{P_s}(M_2) = M_2$, respectively. In particular, b and c are—in a sense—self-supported. Consider the reducts $P_s^{M_1} = \{a \leftarrow;\ b \leftarrow c;\ c \leftarrow b\}$ and $P_s^{M_2} = \{b \leftarrow c;\ c \leftarrow b\}$. We have that $LM(P_s^{M_1}) = \{a\} = M_1$ and $LM(P_s^{M_2}) = \emptyset \neq M_2$, thus M_1 is a stable model, whereas M_2 is just a minimal model, but not a stable one.

Self-supported atoms are in general not desirable, since they can lead to paradoxical scenarios in which true facts are not supported by evidence; a and b from the previous example are indeed *unfounded* w.r.t M_2 in the sense specified below.

Definition 13 ([122]). *Given a program P, a set $U \subseteq HB_P$ is an* unfounded set *of P relative to an interpretation I, if for every $a \in U$ and every $r \in ground(P)$ with $H(r) = a$, either*

1. *There is some atom b appearing as positive literal in the body of r which is such that either $b \notin I$ or $b \in U$, or*
2. *There is some atom b appearing as negative literal in the body of r such that $b \in I$.*

For normal programs there exists the greatest unfounded set of P relative to I, denoted by $U_P(I)$.

Intuitively, if I is compatible with P, then all atoms in $U_P(I)$ can be safely switched to false and the resulting interpretation is still compatible with P. Assuming I as a set of true facts, there is no rule in P that can justify an atom $a \in U$ becoming true.

An interpretation I is called *unfounded-free*, if $U_P(I) = \{\}$.[7]

The notion of unfounded set extends the notion of "non-supportedness" by implicitly forbidding support of an atom by an atom which is unfounded. For gaining "foundedness" by M an atom $a \in M$ necessitates support by a rule whose body is made true by founded atoms only (not belonging to the unfounded set at hand).

Theorem 7 (implicit in [79]). *Given a program P, a model M of P is stable iff M is* unfounded-free.

Example 23. If we take P_u again in consideration

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), \text{not } husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), \text{not } single(dilbert). \qquad (r_2)$$

And the four following "candidate" interpretations:

– $M_1 = \{man(dilbert), single(dilbert)\}$,
– $M_2 = \{man(dilbert), husband(dilbert)\}$,
– $M_3 = \{man(dilbert), single(dilbert), husband(dilbert)\}$
– $M_4 = \{man(dilbert)\}$,

[7] Note that, for more general classes of programs than normal programs (e.g., disjunctive program as later defined in Section 5.3), $U_P(I)$ is undefined. More generally, we can then say that I is unfounded-free, if there is no (non-empty) subset of I which is an unfounded set.

One can observe that M_3 has the greatest unfounded set $U_{P_u}(M_3) = \{single(dilbert), husband(dilbert)\}$: assuming M_3 as a set of "true" facts, there is indeed no rule which could make atoms in $U_{P_u}(M_3)$ true. M_3 is thus not unfounded-free. Note that M_4 is not a model at all, since r_1 and r_2 are not satisfied.

Example 24. Note that the minimal model $M_2 = \{b, c\}$ of P_s is not unfounded free: indeed $U_{P_s}(M_2) = \{b, c\}$.

Reasoning from stable models. Since a logic program P might have no, one, or multiple stable models, the question is how inference from P should be defined. With respect to a particular stable model M, a ground atom a is considered to be true (denoted $M \models a$), if $a \in M$, and false, if $a \notin M$. This is usually extended to inference from all stable models of P in two dual modes, as mentioned already in Section 3.2:

Brave Reasoning. An atom a is a *brave* (or *credulous*) consequence of P, denoted $P \models_b a$, if $M \models a$ for some stable model of P;

Cautious Reasoning. An atom a is a *cautious* (or *skeptical*) consequence of P, denoted $P \models_c a$, if $M \models a$ for every stable model of P.

These notions can be extended to propositional combinations of ground atoms in the natural way (where $M \models \neg a$ iff $a \notin M$), and similarly to (combinations of) closed formulas.

Both $\models_b$ and $\models_c$ are *nonmonotonic*, as adding further rules to P might invalidate a conclusion.

Example 25. If we reconsider the program P_m in Example 20, then both $P_m \models_b r(a)$ and $P_m \models_c r(a)$, as $r(a)$ is true in the unique stable model of P_m. However, for $P'_m = P_m \cup \{q(a)\}$, neither $P'_m \models_b r(a)$ nor $P'_m \models_c r(a)$ holds, as $r(a)$ is false in the single stable model $\{p(a), q(a)\}$ of P'_m.

From this example, one might believe that the nonmonotonic behavior of inference is due to the fact that we added some fact ($q(a)$) that was missing before, but that this would not happen if the fact were already a consequence; that is, that inference satisfies *cautious monotonicity*:

– If $P \models_x a$ and $P \models_x b$, then $P \cup \{a\} \models_x b$.

where $x \in \{b, c\}$. This property is obviously fulfilled for classical inference $\models$ in place of $\models_x$. However, it does not hold for cautious reasoning under stable semantics.

Proposition 2. *In general, $P \models_c a$ and $P \models_c b$ does not imply that $P \cup \{a\} \models_c b$.*

In fact, the property fails even if P has a single stable model. For example, consider the program $P = \{b \leftarrow \text{not } c;\ c \leftarrow \text{not } b;\ a \leftarrow \text{not } a;\ a \leftarrow b\}$. This program has the single stable model $M = \{a, b\}$, and thus $P \models_c a$ and $P \models_c b$. However, the program $P \cup \{a\}$ has another stable model, viz. $N = \{a, c\}$, and thus $P \cup \{a\} \not\models_c b$. The property is, however, true for brave reasoning.

Similarly then, also the stronger property of *cumulativity* fails:

– If $P \models_x a$, then $P \models_x b$ iff $P \cup \{a\} \models_x b$.

That is, by adding consequences as "lemmas," we might change the set of conclusions that can be drawn (which is not the case for classical inference $\models$). In fact, this property also fails for brave reasoning, as shown by the above examples (e.g., $P \cup \{a\} \models_b c$ while $P \not\models_b c$).

In conclusion, care is needed when arguing about how rules in a program compute truth values for atoms under stable semantics. As long as atoms do not depend on negation through cycles, i.e., in the stratified part of a program, adding atoms that are computed true as facts does not change the semantics. Fortunately, this can be generalized to settings where a program can be split into an "lower' and an "upper" part where the former informally provides input to the latter in a modular way [86]. In other cases, one has to carefully examine the effects of adding atoms—in an unfounded way—as facts. More about properties of consequences from stable models can be found e.g. in [61].

4.4 Computational Properties

There are many computational tasks related to logic programs under stable model semantics: one might want to check if a given program P is consistent (that is, it admits at least one stable model), or to compute one, or all, of its models. Also it can be of interest to determine truth of a given query Q under brave or cautious reasoning. We briefly focus here on the problem CONS of deciding whether a given input program P has some stable model, that is, deciding the consistency of P under stable model semantics. The computational complexity of CONS has direct impact on other related problems, thus giving an indication of the complexity of other related problems. For instance, evidence of consistency can be given by computing one stable model.

It turns out that assessing consistency of a ground program P is in general NP-complete.

Theorem 8 ([91]). *The problem* CONS *of deciding whether a given ground program P has some stable model is* NP*-complete.*[8]

Intuitively, this result can be justified by thinking of a simple nondeterministic algorithm for checking the existence of a stable model for P. For showing that CONS is in NP one can: (i) guess a candidate stable model M; (ii) check in polynomial time if M is stable (e.g. by verifying $U_P(M) = \{\}$). Also, one can show that it is possible to build a program P_ϕ, having a stable model iff a given propositional formula ϕ in CNF is true (where P_ϕ is of size at most polynomially higher than the size of ϕ).

However, computational complexity might change depending on allowed extensions (disjunction, presence of function symbols, etc.):

– For "not"-free programs and stratified programs, CONS can be solved in polynomial time (in fact, solvable in linear time);

[8] Recall that NP is the class of problems solvable in polynomial time on a non-deterministic Turing machine [102].

- For programs with variables but not function symbols, CONS has exponentially higher complexity (NEXP-complete);
- For non-ground, arbitrary programs (allowing functional terms), CONS is undecidable. There are however known syntactic conditions on the usage of function symbols which retain complexity in 2-EXP [116,51] resp. 2-NEXP [119,20].[9]

It is important to note the dramatic change in complexity when P is non-ground. This should not be surprising if one considers that, usually, $grnd(P)$ is exponentially bigger than P.

Example 26. Given the rule r_g

$$r(X_1,\ldots,X_k) \leftarrow h(a,b), c_1(X_1),\ldots,c_k(X_k)$$

one can easily observe that $|grnd(r_g)| = O(2^k)$.

In particular one can observe that the size of a grounded program can be exponentially bigger than its original non-ground counterpart if k is allowed to vary, that is, if programs can have arbitrarily long rules, and arbitrarily large arities. This might not be the case if a bound on such parameters is given (see e.g. [35]). Also, one might wonder why the introduction of function symbols makes CONS undecidable. One can easily see that, in this setting, it is possible to have stable models of infinite size:

Example 27. Consider the program P_f:

$$\begin{aligned} p(a).\\ p(f(X)) \leftarrow p(X). \end{aligned}$$

We can observe that $grnd(P_f) = \{p(a), p(f(a)) \leftarrow p(a), p(f(f(a))) \leftarrow p(f(a)),\ldots\}$ is infinite, as well as its unique stable model $M_{inf} = \{p(a), p(f(a)), p(f(f(a)),\ldots\}$.

It is thus not surprising that for non-ground programs, admitting functions symbols, CONS and other related reasoning problems become as difficult as deciding the termination of a Turing machine on a given input.[10]

5 Extensions

In the above sections, we have dealt with the motivation and history of answer set programming, and described syntax and semantics of gradually increasing expressive program classes. In particular, we looked into the class of normal logic programs under stable models semantics and showed their alluring semantic properties. But, so far, we did not touch upon the full area of answer set programming. In this section, we will approach the main topic of this chapter and show more syntactic extensions of normal

[9] A decision problem is in 2EXP (2NEXP) time, if it can be solved by a (non-)deterministic Turing Machine in time $O(2^{2^{p(n)}})$, where $p(\cdot)$ is a polynomial and n is the size of the input instance.

[10] The reader can find in [25] a thorough collection of results regarding computational complexity of logic programming under various semantics including the stable models semantics.

logic programs and define their semantics, which, eventually, brings us to the answer set semantics.

We now turn our attention to three particular extensions of normal logic programs that lead to the notion of Answer Set Programming: (i) (integrity) constraints (rules with empty head) like

$$\leftarrow \mathit{edge}(X,Y), \mathit{red}(X), \mathit{red}(Y) \ , \tag{9}$$

which forces that adjacent nodes in a graph are not allowed to have the colour red; (ii) strong (or "classical") negation in atoms, e.g., $-\mathit{single}(\mathit{dilbert})$ (Dilbert is *known* not to be a single); and (iii) disjunctive rules, i.e., allowing for disjunctions in rule heads like in

$$\mathit{female}(X) \vee \mathit{male}(X) \leftarrow \mathit{person}(X) \ ,$$

which intuitively means that persons are either female or male. For many crucial knowledge representation tasks, these extensions are not only desirable, but also necessary for succinct encodings of problems. Programs that permit strong negation are also called *Extended Logic Programs* (ELP). If ELPs additionally allow for disjunctive rules, we obtain the class of disjunctive ELPs, which are also called *Disjunctive Logic Programs* (DLP).

Next, we will look into these important extensions in more detail and then provide syntax and semantics of ELPs and DLPs.

5.1 Constraints

Integrity constraints check admissibility of models, possibly using auxiliary predicates defined by normal stratified rules. For instance, the constraint rule (9) can be equally well expressed as the "killing clause"

$$\mathit{falsity} \leftarrow \mathrm{not}\ \mathit{falsity}, \mathit{edge}(X,Y), \mathit{red}(X), \mathit{red}(Y) \ , \tag{10}$$

where $\mathit{falsity}$ is a fresh propositional atom. Now, if there is an interpretation I for a program with the constraint (9) such that I contains $edge(a,b)$, $red(a)$, and $red(b)$, but $falsity \notin I$, then (10) is applicable and forces $falsity$ to be true. But then, I cannot be a model for our program, as $falsity$ is false in I. This means that (10) "kills" all models that do not satisfy the constraint (9).

5.2 Strong Negation

In Section 3, we have defined normal logic programs, i.e., logic programs that allow for weak negation in rule bodies. The intuitive meaning of $\mathrm{not}\ a$ is that "a cannot be proved (derived) using rules," and that a is false by default (or *believed* to be false). But this is different from (provably) *knowing* that a is false, which is expressed by $\neg a$; in ASP, one also writes $-a$ for this.

Example 28 (by John McCarthy). Consider an agent A with the following task: "At a railroad crossing, cross the rails if no train approaches." We may encode this scenario using one of the following two rules:

$$walk \leftarrow at(A, L), crossing(L), \text{not } train\_approaches(L). \quad (11)$$
$$walk \leftarrow at(A, L), crossing(L), -train\_approaches(L). \quad (12)$$

In the following, let us assume that A is at some crossing L.

If we take (11) as encoding for the railroad-crossing task, and A cannot infer from her beliefs that $train\_approaches(L)$ is true, then A will conclude to walk even though A cannot be sure that there is no approaching train: her beliefs might not represent the state of the world completely. Now, if we take (12) as the encoding, A will only $walk$ if she can prove that there is no approaching train.

In (11), an update to A's knowledge can lead to revised conclusions; if we add $train\_approaches(L)$, then A will refuse to walk. This is the typical behavior of non-monotonic rules like (11), but may not be desired in critical situations like crossing a railroad, as an approaching train, which has not been perceived by A yet, might cause devastating effects on the agent. From this point of view, the rule (12) employing strong negation is preferable.

There are several ways to express negative knowledge using strongly negated atoms. One way is to explicitly state them as facts in a knowledge base. For instance, the fact $-broken(battery)$ expresses that a battery is definitely not broken. If this knowledge base concludes in a different rule that $broken(battery)$ holds, then we face inconsistency, and this causes to vanish all models of that particular knowledge base.

Another useful application for strong negation (in combination with weak negation) is to express *default rules*. For example, we can express that "a bird flies by default" with the rule $flies(X) \leftarrow bird(X), \text{not } -flies(X)$.

Extended Logic Programs. Adding strong negation to normal logic programs leads to the so called *extended logic programs*.

Definition 14. *An* extended logic program (ELP) *is a finite set of rules*

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \qquad (n, m \geq 0) \quad (13)$$

where a and all b_i, c_j are atoms or strongly negated atoms in a first-order language L.

The semantics of ELPs can be defined in different ways, either genuinely by considering sets of ground literals rather than sets of atoms as basis, as done in [64], or by a simple reduction to normal logic programs that compiles strong negation away; we follow here for simplicity the latter. To this end, we

– view negative literals "$-p(\boldsymbol{X})$" as atoms with fresh predicate symbols $-p$, for each atom $p(\boldsymbol{X})$;
– add the clause

$$falsity \leftarrow \text{not } falsity, p(\boldsymbol{X}), -p(\boldsymbol{X}) \quad (14)$$

to P (this prevents that $p(\boldsymbol{X})$ and $-p(\boldsymbol{X})$ are true at the same time); and
– select the stable models of the resulting program P'. These are called *answer sets* of P.

Note that extended logic programs have similar properties as normal logic programs under stable models semantics. For a ground atom a, constraint (14) prevents that both a and $-a$ are contained in answer sets. One takes a *three-valued view* on this: an atom may be true, false, or *undefined* (i.e., we *don't know* if the atom is true or false). This contrasts with the two-valued view of stable models of a normal logic program, in which an atom a is either true (if a is in the model) or false (if a is not on the model, in the spirit of Reiter's Closed World Assumption [111]).[11]

The use of strong negation may cause inconsistency, even if a program does not have weak negation. For example, take the program P

$$\begin{aligned} &\mathit{true}. \\ \mathit{trivial} &\leftarrow \mathit{true}. \\ a &\leftarrow \mathit{true}. \\ -a &\leftarrow \mathit{true}. \end{aligned}$$

which derives both a and $-a$. The constraint (14) prevents that P has answer sets, thus P is inconsistent. However, this inconsistency is of a different quality than the one cause by default negation (cf. Example 17).

Example 29. The next program is a knowledge base for determining if one should query the *science citation index* (sci) or the citeseer database:

$$\begin{aligned} \mathit{up}(S) &\leftarrow \mathit{website}(S), \text{not } -\mathit{up}(S). && (r_1) \\ -\mathit{query}(S) &\leftarrow -\mathit{up}(S). && (r_2) \\ \mathit{query}(\mathit{sci}) &\leftarrow \text{not } -\mathit{query}(\mathit{sci}), \mathit{up}(\mathit{sci}). && (r_3) \\ \mathit{query}(\mathit{citeseer}) &\leftarrow \text{not } -\mathit{query}(\mathit{citeseer}), -\mathit{up}(\mathit{sci}), \mathit{up}(\mathit{citeseer}). && (r_4) \\ \mathit{flag\_error} &\leftarrow -\mathit{up}(\mathit{sci}),\ -\mathit{up}(\mathit{citeseer}). && (r_5) \\ \mathit{website}(\mathit{sci}). &\qquad \mathit{website}(\mathit{citeseer}). \end{aligned}$$

In rule (r_1), we define that websites are up by default, and (r_2) encodes that a website known to be not up should not be queried. The rules (r_3) and (r_4) give a preference on the websites: whenever we cannot prove that sci is not usable and sci is available, then we should query the science citation index, but we should only query $\mathit{citeseer}$ if it is available for querying and sci is down. In (r_5), we simply raise an error-flag whenever both websites are down.

The single answer set of this program is

$$M = \{\mathit{website}(\mathit{sci}), \mathit{website}(\mathit{citeseer}), \mathit{up}(\mathit{sci}), \mathit{up}(\mathit{citeseer}), \mathit{query}(\mathit{sci})\}\ ,$$

whose intuitive meaning is that we should query the science citation index, even though $\mathit{citeseer}$ is up and running.

[11] The answer sets of an ELP P without strong negation coincide with the stable models of P, and thus the terms are often used interchangeably (confusing two- vs three-valuedness).

If we add to our knowledge base the rule

$$-query(S) \leftarrow \text{not } query(S), -reliable(S) \qquad (r_6)$$

and the facts that sci is down and $citeseer$ is unreliable,

$$-up(sci) \text{ and } -reliable(citeseer) \ ,$$

we can witness a different behavior. Intuitively, (r_6) creates a nondeterminism in our program, as for websites S that are known to be unreliable we can infer $-query(S)$, provided that we cannot prove $query(S)$. But rules (r_3) and (r_4) gives us similar knowledge, except with unlike signs: we can infer a positive fact $query(S)$ given that we cannot prove $-query(S)$. To resolve this conflicting views, we obtain for our knowledge base under answer set semantics exactly two answer sets, with each intuitively describe the corresponding alternative view on our site selection problem:

- $M_1 = \{website(sci), website(citeseer), -up(sci),$
  $up(citeseer), -reliable(citeseer), -query(sci), query(citeseer)\}$, and
- $M_2 = \{website(sci), website(citeseer), -up(sci),$
  $up(citeseer), -reliable(citeseer), -query(sci), -query(citeseer)\}$,

i.e., in M_1 we have chosen to query $citeseer$ and in M_2 we conclude that we should not query $citeseer$, thus querying no website at all.

Relationship to Reiter's Default Logic. It has been noted already in [64] that ELPs are closely related to Reiter's famous Default Logic [112]. For an ELP clause C of form (13), consider the corresponding default

$$d(C) = \frac{b_1 \wedge \cdots \wedge b_m : \neg.c_1, \ldots, \neg.c_n}{a} \ ,$$

where $\neg.c_i$ is the opposite of c_i (i.e., $\neg.a = \neg a$ and $\neg.\neg a = a$).

Theorem 9. *Let P be an extended logic program and let $T = (\emptyset, \{d(C) \mid C \in P\})$ be the corresponding default theory. Then, M is an answer set of P if and only if $E = Cn(M)$ is a consistent default extension of T.*

Thus, extended logic programs under answer set semantics can be regarded as a fragment of default logic.

5.3 Disjunction

The next extension to normal logic programs is disjunctions in rule heads. The use of disjunction is natural to express indefinite knowledge. For instance, the rule

$$female(X) \vee male(X) \leftarrow person(X)$$

expresses that all persons are either female or male. Another example is the disjunctive fact

$$broken(left\_hand, tom) \vee broken(right\_hand, tom) \leftarrow \ ,$$

which expresses that tom has a broken arm, but it is unknown whether the left or the right hand is broken.

Disjunctive information is a natural extension for expressing a "guess" and to create non-determinism in logic programs, like in the rule

$$ok(C) \vee -ok(C) \leftarrow component(C) \ ,$$

which states that a component may be in a working condition or not working at all, and in each of the alternative states of the component, we might have to take different actions for solving our problem.

The semantics of disjunctive rules is such that we conclude one of the alternatives to be true (the minimality principle).

In the next example, we look at different disjunctive programs, and the models they admit.

Example 30. Disjunction is *minimal*, i.e., from a rule, we usually infer only a single atom "at a time." The single rule program

$$a \vee b \vee c \leftarrow \tag{15}$$

has three minimal models: $\{a\}$, $\{b\}$, and $\{c\}$. There exist no smaller models for (15), since $\emptyset$ is not a model. The interpretation $I = \{a, b\}$ for instance is a model of this program, but both $\{a\}$ and $\{b\}$ are smaller than I and satisfy (15), hence I is not a minimal model.

If we take a closer look into the minimal models of disjunctive programs, we observe that they are actually *subset minimal*. Take, for instance, the program

$$a \vee b \leftarrow \tag{16}$$

$$a \vee c \leftarrow \tag{17}$$

This program has two minimal models: $\{a\}$ and $\{b, c\}$. The interpretation $J = \{a, b, c\}$ is a model for both (16) and (17), hence it is a model of the program. But J is a proper superset of $\{b, c\}$, thus J is not a minimal model. Note that $\{a\}$ is the only singleton minimal model, as both $\{b\}$ and $\{c\}$ do not satisfy the program.

In a similar vein, the program

$$a \vee b \leftarrow \tag{18}$$

$$a \leftarrow b \tag{19}$$

has the single minimal model $\{a\}$, as the model $\{a, b\}$ is not minimal with respect to set inclusion. The interpretation $\{b\}$ is not a model; it satisfies rule (18), but it is not a model for (19).

Note that disjunction should not be understood as *exclusive*. Take program

$$a \vee b \leftarrow \tag{20}$$

$$b \vee c \leftarrow \tag{21}$$

$$a \vee c \leftarrow \tag{22}$$

which has three minimal models $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$. Each of the three minimal models are not contained in the other, but the intersection of any two of the minimal models is nonempty.

Let us next consider the use for disjunctive rules vs. unstratified negation. Going back to our Dilbert scenario, the program

$$man(dilbert). \tag{23}$$
$$single(X) \leftarrow man(X), \text{not } husband(X). \tag{24}$$
$$husband(X) \leftarrow man(X), \text{not } single(X). \tag{25}$$

which expresses that a man is either a single or a husband, is equivalent to the disjunctive program

$$man(dilbert). \tag{26}$$
$$single(X) \lor husband(X) \leftarrow man(X). \tag{27}$$

Here, the use of disjunction is more intuitive. In fact, one can see the rule (27) resulting from (24) resp. (25) by "shifting" the negated literal not $husband(X)$ (resp. not $single(X)$) to the head (classically, the clauses are equivalent). While such shifting works in this example, as well as under certain syntactic conditions (like headcycle-freeness) [13], we note that in general, disjunctive rule heads are not syntactic sugar for unstratified negation; this is also evidenced by complexity results provided in Section 5.3.

Extended Logic Programs with Disjunctions. The extension of ELPs with disjunctive rule heads leads to the class of extended disjunctive logic programs in [64].

Definition 15. *A* extended disjunctive logic program *(EDLP) is a finite set of rules*

$$a_1 \lor \cdots \lor a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \qquad (k, m, n \geq 0)$$

where all a_i, b_j, and c_l are atoms or strongly negated atoms.

The semantics for an EDLP can be defined similarly as for an extended logic program, with the only difference being that instead of choosing a stable model M of P (i.e., M is the least model of the reduct P^M), we define an *answer set* M of an EDLP P as a *minimal model* M of the reduct P^M, since multiple minimal models of P^M might exist.

Example 31. Consider the program P:

$$man(dilbert).$$
$$single(X) \lor husband(X) \leftarrow man(X).$$

There are two answer sets for P:

– $M_1 = \{man(dilbert), single(dilbert)\}$, and
– $M_2 = \{man(dilbert), husband(dilbert)\}$.

Please note that P is "not"-free (positive), hence the reduct $grnd(P)^M = grnd(P)$ for every interpretation M.

It is worth mentioning here that answer sets of EDLPs can also be nicely defined in equilibrium logic [103,104], which is a non-monotonic version of the logic of here and there, an intermediate logic between classical logic and intuitionistic logic. This logic is well-suited to capture not only EDLPs, but also other extensions of normal logic programs.

Semantic Properties of Disjunctive ELPs. The extensions of normal logic programs to ELPs and DLPs considered in this section inherit most of the alluring properties of stable models, which have been shown in Section 4.3.

We now define Herbrand interpretations to EDLPs. Since an extended logic program may contain atoms under classical negation, an interpretation for EDLPs may also contain strongly negated ground atoms, i.e., literals of form a or $-a$. But this means that an interpretation can be inconsistent if it contains both a and $-a$. In [64], the inconsistent answer set has been defined as the interpretation which contains all possible atoms and their strongly negated counterparts. For our purposes, we deal only with consistent interpretations and thus disregard the inconsistent answer set. We define models as follows.

Definition 16. *A interpretation I is a* model *of*

- *a ground clause $C : a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$, denoted $I \models C$, if either $\{b_1, \ldots, b_m\} \not\subseteq I$ or $\{a_1, \ldots, a_k, c_1, \ldots, c_n\} \cap I \neq \emptyset$;*
- *a clause C, denoted $I \models C$, if $I \models C'$ for every $C' \in grnd(C)$;*
- *a program P, denoted $I \models P$, if $I \models C$ for every clause C in P.*

The above definition takes all of our extensions into account: (i) constraints do not have head literals, hence $k = 0$ and only the body part (the b_i, c_j) is taken into account; (ii) rules with strong negation are considered by viewing all a_i, b_j, c_l in I as classical literals; as well as (iii) disjunctive rules (where $n > 1$), with the meaning that if the rule body is satisfied, at least some literal a_i, $1 \leq i \leq n$, must be true. In a sense, such interpretations represent three-valued states: a ground atom a is regarded *true* in I, if $a \in I$, while a is regarded *false* in I, if $-a \in I$; of neither a nor $-a$ is contained in I, then a is *unknown* in I.

Similar to stable models of normal logic programs, a (disjunctive) ELP P may have no, one, or multiple answer sets, which are models of P, and, in fact, minimal models of P.

Theorem 10. *Let P be a (disjunctive) ELP and M be an answer set of P.*

1. *M is a model of P.*
2. *M is a minimal model of P.*

Hence, just like least models of positive programs and stable models of normal programs, an answer set satisfies all rules of an EDLP. Moreover, an answer set is a minimal model of the program, which intuitively means that it contains only the absolutely necessary bare minimum of facts in order to satisfy a program.

Corollary 2. *If M_1 and M_2 are two different answer sets of P then $M_1 \not\subseteq M_2$ and $M_2 \not\subseteq M_1$.*

Similarly to stable models for normal logic programs, one can define unfounded sets for answer sets of of EDLPs to address the problem of self-supported literals. Leone et al. [79] did this for programs without strong negation.

Definition 17 ([79]). *Given an EDLP P without strong negation and an interpretation I, a set $U \subseteq HB_P$ is an* unfounded set *of P relative to an interpretation I, if for every $a \in U$ and every $r \in ground(P)$ such that a appears in the head of r, at least one of the conditions hold:*

1. *There is a literal b appearing in the positive body of r such that either $b \notin I$ or $b \in U$;*
2. *There is a literal b appearing in the negative body of r such that $b \in I$; or*
3. *There is a literal b appearing in the head of r such that $b \notin U$ and $b \in I$.*

Unlike for normal logic programs, we cannot guarantee the existence of a greatest unfounded set for disjunctive programs relative to an interpretation. But there exist interpretations for an EDLP, where the existence of a greatest unfounded set is guaranteed: the unfounded-free interpretations. We call an interpretation I for an EDLP P *unfounded-free*, if $I \cap U = \emptyset$ for each unfounded set U for P w.r.t I.

Theorem 11 ([79]). *Given an EDLP program P without strong negation, an interpretation M of P is an answer set iff M is* unfounded-free.

The DLV system heavily relies on unfounded sets as its underlying principle to build the answer sets of EDLPs (see Section 7.2 for more details).

Note that the notion of unfounded set is easily extended to answer sets of EDLPs with strong negation, by adding the respective constraints; [48] defines unfounded sets directly in Equilibrium Logic.

A recent development in the ASP area is a syntactic counterpart of unfounded sets: *loop formulas* [87,75,74]. These formulas have been conceived as a way to transform logic programs under stable and answer set semantics to propositional theories, and let standard SAT solvers perform the task of computing the stable models of these extended theories. In a nutshell, this translation uses Clark's completion [23] for logic programs to create a propositional theory and augment this theory by additional loop formulas, which guarantee that this theory admits only stable models. Note that in general there can be exponentially many loop formulas for a given EDLP [85].

We end this section by looking into reasoning with answer sets, which is defined just as reasoning with stable models: a classical literal a is a (i) *brave (credulous) consequence* of program P, $P \models_b a$, iff $M \models_b a$ for some answer set M of P; and (ii) *cautious (skeptical) consequence* of a program P, $P \models_c a$, iff $M \models_c a$ for all answer sets M of P. The behaviour with respect to properties like cautious monotonicity and cumulativity is then similar as in the disjunction-free case.

Computational Properties of Disjunctive ELPs. Similar to normal logic programs under stable model semantics, EDLPs under answer set semantics have many interesting computational tasks, and a particular one is testing whether a program P is consistent, i.e., whether P has some answer set. Here, we restrict our attention to the consistency problem, give complexity results for various classes of EDLPs, and briefly sketch proofs or give ideas how such a proof can look like. Let us start with the general case.

Theorem 12 ([39]). *Deciding whether a given ground disjunctive program P has some answer set is Σ_2^p-complete in general.*

Recall that $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ is the class of problems decidable in polynomial time on a nondeterministic Turing machine with an oracle for solving problems in NP [117].

The membership of consistency of disjunctive ELPs can be shown by the following argument: we first guess an answer set M for a program P, and verifying whether M

is a minimal model of P^M is in co-NP (note that P^M can be computed in polynomial time), thus decidable with one call to an NP-oracle.

The intuition for the hardness part is as follows: we have to create a reduction from validity of a quantified Boolean formula of the form $\exists X \forall Y E(X,Y)$ to an EDLP P, where $E(X,Y)$ is in disjunctive normal form and X and Y are the (lists of) variables occurring in E. For a detailed proof, we refer the reader to [39].

But there exist subclasses of EDLPs with lower computational complexity. For instance, testing whether a strictly positive disjunctive ELPs has an answer set is easy, since each positive program has a model and the Gelfond-Lifschitz reduct does not change the given program.

For a ground DLP P with constraints, but without "not" in the rule bodies, deciding whether P has some answer set is NP-complete. Hardness can be shown by a reduction from SAT: given a propositional CNF-formula ϕ, we transform ϕ into a positive disjunctive program P with constraints by adding rules $a \vee \bar{a} \leftarrow$ for each atom a in ϕ and adding the "negation" C' of each clause C in ϕ as a rule of form $u \leftarrow C'$ to P, where u is a fresh symbol (e.g., for a clause $a \vee b \vee \neg c$ we add the rule $u \leftarrow \bar{a}, \bar{b}, c$), and finally add the constraint $\leftarrow u$; then, ϕ is satisfiable iff P has an answer set. Membership can be shown by guessing an interpretation for the positive part of the program, and checking if the interpretation is a minimal model of the positive part and is compliant with the constraints in polynomial time.

Other classes of EDLPs exist which obtain lower complexity, for instance, deciding consistency of headcycle-free EDLPs [13] is also NP-complete.

Similarly to normal logic program, we obtain an exponential blowup for nonground EDLPs compared to the propositional case. In particular, verifying whether a nonground EDLP has some answer set is $\mathrm{NEXP}^{\mathrm{NP}}$-complete, i.e., complete for the class of problems that run in exponential time on a nondeterministic Turing machine and have access to an NP-oracle (see also [25]). If function symbols are allowed, the complexity does not increase through disjunction in general (cf. [25]); syntactic restrictions are known under which the complexity of deciding consistency stays is 2-EXP-complete [116] and 3-EXP-complete [51], respectively.

6 Answer Set Programming Paradigm

In this section, we now turn to the Answer Set Programming (ASP) paradigm, which emerged from the nonmonotonic Logic Programming area at the end of the 1990s. There were, as already mentioned, several texts in which this paradigm was proposed, [81,82,92,96]; after the LPNMR 1999 conference, a special issue of the AI Journal was edited [62] covering the subject, a dedicated ASP workshop series started in 2001 [108]. The textbook by Baral [10] was then a further step to disseminate this approach.

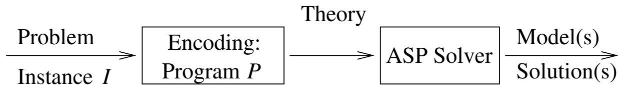


Fig. 6. Encoding of problems in ASP

Let us first start with more motivation by outlining the general idea behind answer set programming: given an instance of a (search) problem I and its corresponding representation in form of a logic program P, we may perceive the *models of* P as *solutions for* I. That is, in ASP, we view problem solving tasks as computing the models of their matching encoded programs. This view gives rise to a general strategy for implementing any kind of problem solving task, shown graphically in Figure 6:

1. we *encode* our problem instance I as a (nonmonotonic) logic program P, such that solutions of I are represented by models of P; and then
2. *compute* some model M of P, by using an AS solver of our choice; and finally
3. *extract* a solution for I from M.

We may vary this strategy by allowing to compute more than one solution, which intuitively corresponds to obtaining multiple or even all solutions for our problem instance I.

This method has been successfully applied to numerous problems in a range of areas; an incomplete list is

– diagnosis
– information integration
– constraint satisfaction
– reasoning about actions (including planning)
– routing, and scheduling
– phylogeny construction
– security analysis
– configuration
– computer-aided verification
– health care
– biomedicine and biology
– Semantic Web
– knowledge management
– text mining and classification
– question answering

The survey [124] is a source for specific applications, some of which can be viewed in an online showcase collection.[12]

We illustrate the ASP approach on the problem of computing legal 3-colorings of a graph.

Example 32. Let $G = (V, E)$ be a graph with nodes $V = \{a, b, c, d\}$ and edges $E = \{(a, b), (b, c), (c, a), (a, d)\}$, which constitutes our problem instance I. We can encode the legal three colorings of G into answer sets of a logic program P as follows. For each node n, we have atoms b_n, r_n, and g_n which informally mean that node n is colored blue, red, and green, respectively. Then we set up the following rules. For each node $n \in V$,

[12] http://www.kr.tuwien.ac.at/research/projects/WASP/showcase.html

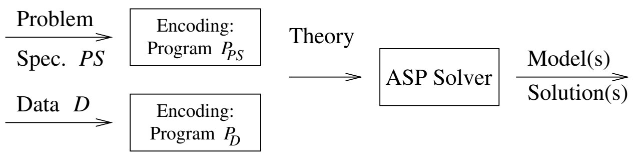


Fig. 7. Uniform problem encoding in ASP

$$\begin{aligned}
b_n &\leftarrow \text{not } r_n, \text{not } g_n. \\
r_n &\leftarrow \text{not } b_n, \text{not } g_n. \\
g_n &\leftarrow \text{not } r_n, \text{not } b_n.
\end{aligned}$$

and for each edge (n, n') in E, the constraints

$$\begin{aligned}
&\leftarrow b_n, b_{n'}. \\
&\leftarrow r_n, r_{n'}. \\
&\leftarrow g_n, g_{n'}.
\end{aligned}$$

Then, the answer sets of P encode 1-1 the legal 3-colorings of G. Informally, the rules for $n \in V$ assign one of the three colors to n, and the constraints for (n, n') check that adjacent nodes do not have the same color. Equally well, we can replace the three rules for $n \in V$ by the single (and perhaps more intuitive) rule

$$b_n \vee r_n \vee g_n \leftarrow .$$

This problem solving strategy is closely related to similar approaches like SAT-solving, where the problem instance is encoded onto the (classical) models of a propositional formula of clause set. It is because of this that some authors refer to Answer Set Programming as the more general paradigm in which a problem is encoded into the models of a logical theory, and consider the usage of nonmonotonic logic programs as theory as a particular instance of this paradigm. We prefer here, however, to reserve the term ASP for the setting with nonmonotonic logic programs under the answer set semantics itself.

Compared to SAT solving, ASP provides features that are not available there. For example, the transitive closure of a given graph G (and its complement) is expressible within an answer set, which is cumbersome in classical propositional logic. Here, one can exploit negation as failure fruitfully. Furthermore, ASP offers many constructs besides negation as failure, and, importantly, allows also problem descriptions with predicates and variables. This can be utilized for generic problem solving where the specification of solutions (the "logic" PS) is separated from the concrete instance of the problem at hand (the "data" D, usually given as facts); see Figure 7.

Example 33. In the graph 3-coloring problem, assuming that $G = (V, E)$ is stored using facts $node(n)$ for each $n \in V$ and $edge(n, n')$ for each $(n, n') \in E$, which gives the data D, the generic specification of solutions PS can be given by the following rules:

$$\begin{array}{l} b(X) \leftarrow node(X), \text{not } r(X), \text{not } g(X). \\ r(X) \leftarrow node(X), \text{not } b(X), \text{not } g(X). \\ g(X) \leftarrow node(X), \text{not } r(X), \text{not } b(X). \end{array}$$

and the constraints

$$\begin{array}{l} \leftarrow b(X), b(Y), edge(X,Y). \\ \leftarrow r(X), r(Y), edge(X,Y). \\ \leftarrow g(X), g(Y), edge(X,Y). \end{array}$$

Similarly as above, we can use the single disjunctive rule

$$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$$

instead of the three unstratified rules defining $b(X)$, $r(X)$, and $g(X)$. Then, the answer sets of $PS \cup D$ correspond to the legal 3-colorings of G.

The efficient evaluation of ASP programs requires the integration of techniques from the areas of Knowledge Representation, Database, and Search, as language constructs and features need to be handled (possibly by compiling them away), (larger) input volumes of data need to be processed, and nondeterminism as it occurs with unstratified negation has to handled with search.

In the rest of this section, we will briefly discuss some declarative programming techniques that are used in ASP. There is a variety of "design patterns" which depend on the type of problem to be solved and the language elements that are used or needed. We discuss here the use of four techniques: (i) double negation as a technique to compute maximal elements in a set, which can done using stratified negation; (ii) a general guess and check methodology which uses unstratified negation or disjunction to generate and prune solutions candidates; (iii) an advanced technique called *saturation*, which can be used in disjunctive logic programming to test properties of various subsets of a set within an answer set, and is essential to solve "hard" problems there (cf. [39,36]); (iv) in combination with this, iteration over a set to test whether a property holds for all elements without the use of negation.

6.1 Use of Double Negation

The first technique which we look at is the use of double negation. In classical logic $\neg\neg A \equiv A$, i.e., double negation can be canceled. This can be similarly exploited in ASP to define a predicate $p(X)$ in terms of its complement $-p(x)$, and is particularly attractive if $-p(x)$ can be defined easily. For example, one can avoid counting and arithmetic for determining the maximum in a (finite) set of numbers.

Example 34. Suppose the data about employees of a company and their salaries are stored as facts $empl(N,S)$ in the data D, where N is the name and S the salary of an employee. Then the maximum salary, $s^* = \max\{s \mid empl(e,s) \in D\}$, is determined by the following simple ASP program:

$$\begin{array}{l} \%\text{ salary S is *not* maximal} \\ -max(S) \leftarrow empl(N,S),\ empl(N_1,S_1), S < S_1. \\ \%\text{ double negation} \\ max(S) \leftarrow empl(N,S), \text{not } -max(S). \end{array}$$

Example 35. For a little more involved example where this technique can be used successfully, consider the problem of computing the greatest common divisor (gcd) of two natural numbers $n, m > 0$; recall that the gcd of n and m is the largest integer d^* such that d^* divides both n and m. This problem is a standard example in logic programming and elegant solutions for it can be found in textbooks, which basically implement Euclid's recursive algorithm for it by rules:

$$\begin{array}{l}
\%\text{ base case} \\
gcd(X, X, X) \leftarrow int(X), X > 1. \\
\%\text{ subtract smaller from larger number} \\
gcd(D, X, Y) \leftarrow X < Y, gcd(D, X, Y_1), Y = Y_1 + X. \\
gcd(D, X, Y) \leftarrow X > Y, gcd(D, X_1, Y), X = X_1 + Y.
\end{array}$$

Here, $int(X)$ is a (built in) predicate for natural numbers ≥ 0. While Euclid's algorithm is ingenious, the average programmer will approach the problem by trying to cast the definition into rules. Here, double negation can be used again to single out the maximal common divisor d^* in the predicate $gcd(X, Y, Z)$ given that the common divisors are computed in a predicate $cd(X, Y, Z)$, which in turn can be done easily using a predicate $divisor(X, Y)$ that is defined using simple (built-in) arithmetic. A respective program is the following:

$$\begin{array}{l}
\%\text{ Declare when } D \text{ divides a number } N. \\
divisor(D, N) \leftarrow int(D), int(N), int(M), N = D * M. \\
\%\text{ Declare common divisors} \\
cd(T, N_1, N_2) \leftarrow divisor(T, N_1), divisor(T, N_2). \\
\%\text{ Single out non-maximal common divisors T} \\
-gcd(T, N_1, N_2) \leftarrow cd(T, N_1, N_2), cd(T_1, N_1, N_2), T < T_1. \\
\%\text{ Apply double negation: take non non-maximal divisor} \\
gcd(T, N_1, N_2) \leftarrow cd(T, N_1, N_2), not\ -gcd(T, N_1, N_2).
\end{array}$$

For a similar encoding in Prolog, one has to be careful to define and use $int(X)$ properly in the rules, otherwise the program might not terminate.

Note that the above programs are both stratified and thus have a single answer set over any input data (provided that some answer set exists). We will next consider programs that are geared toward multiple answer sets for capturing problems with multiple solutions.

6.2 The "Guess and Check" Methodology

An important element of ASP is to employ a "Guess and Check" methodology, which is sometimes also called Generate-and-Test [82]. The idea is here to proceed as follows:

1. use nondeterminism that comes with unstratified negation, or equally well with disjunction in rule heads, to create candidate solutions to a problem (program part $\mathcal{G}$), and
2. to check with further rules and/or constraints, whether a solution candidate is proper (program part $\mathcal{C}$). This part may also involve auxiliary predicates, if needed.

From another perspective, the part $\mathcal{G}$ defines the search space, and the part $\mathcal{C}$ prunes illegal branches. A detailed discussion of this paradigm is given in [78,36], and in [78] it is extended by a further component to compute optimal solutions (we will deal with this in Section 7.2 below). We will just briefly illustrate the methodology on a few examples.

Example 36. As a first example, we revisit the 3-colorability problem in Example 33.

$$g(X) \vee r(X) \vee b(X) \leftarrow node(X) \Big\} \textbf{Guess}$$

$$\left.\begin{array}{l} \leftarrow b(X), b(Y) \\ \leftarrow r(X), r(Y) \\ \leftarrow g(X), g(Y). \end{array}\right\} \textbf{Check}$$

The first disjunctive rule constitutes the guessing part $\mathcal{G}$, which generates all possible assignments of one colors to the nodes of the graph, while the three constraints constitute the checking part $\mathcal{C}$.

The next example shows a checking part which uses auxiliary predicates.

Example 37. Recall that for a directed graph $G = (V, E)$, a path $n_0 \rightarrow n_1 \rightarrow \cdots \rightarrow n_k$ in G from a start node $n_0 \in V$ is called a *Hamiltonian path*, if all nodes n_i are distinct and each node in V occurs in the path, i.e., $V = \{n_0, \ldots, n_k\}$. Assume that, as above, the graph G is stored using the predicates $node(X)$ and $edge(X, Y)$, and that a predicate $start(X)$ stores the unique node n_0. Consider the following program:

$$inPath(X,Y) \vee outPath(X,Y) \ \leftarrow \ edge(X,Y). \Big\} \textbf{Guess}$$

$$\left.\begin{array}{l} \leftarrow inPath(X,Y),\ inPath(X,Y_1),\ Y \neq Y_1. \\ \leftarrow inPath(X,Y),\ inPath(X_1,Y),\ X \neq X_1. \\ \leftarrow node(X),\ \text{not}\ reached(X). \end{array}\right\} \textbf{Check}$$

$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inPath(Y,X). \end{array}\right\} \textbf{Auxiliary Predicate}$$

The guessing part $\mathcal{G}$ simple states for each edge of the graph whether it belongs to the path or not. The checking part $\mathcal{C}$ tests whether $inPath$ really constitutes a path in G in which each node occurs only once (which is ensured if there is at most one edge from/to each node), and that all nodes are on the path. For this, the auxiliary predicate $reached(X)$ is used, which expresses that the node X is reached from the starting node. The latter is expressed with two simple recursive rules.

Note that deciding the existence of a Hamiltonian path is, like 3-colorability, NP-complete; a similar SAT encoding would be, due to the reachability check, more cumbersome.

As a final example, we consider a scenario where the checking part is interfering with the guessing part, which shows that the two parts may not always be cleanly separated. In fact, this happens for the elementary task of choosing an element from a set.

Example 38. Suppose departments of a company are stored in a predicate $dept(X)$, and the task is to choose a single department; in general, there will be multiple choices (or none, if $dept$ would be empty). The following program is a simple (yet little elegant) solution to the problem:

$$sel(D) \vee -sel(D) \leftarrow dept(D). \quad \Big\} \textbf{ Guess}$$

$$\left.\begin{array}{r l} & \leftarrow sel(D_1), sel(D_2), D_1 \neq D_2. \\ some\_sel & \leftarrow sel(D) \\ & \leftarrow dept(D), \text{not } some\_sel \end{array}\right\} \textbf{Check}$$

Here, the checking part tests that not more than one department has been chosen, and that at least one is chosen if there are departments (hence, exactly one is chosen).

A more elegant solution is to let the checking part interfere with the guessing part, and to exploit the minimality property of answer sets.

$$\begin{array}{r l l} sel(D) & \leftarrow dept(D), \text{not } -sel(D). & \big\} \textbf{ Guess} \\ -sel(D_1) & \leftarrow dept(D_1), sel(D_2), D_1 \neq D_2 & \big\} \textbf{ Check} \end{array}$$

The guessing rules informally states that, by default, a department D is chosen. The checking rule says that if some department is chosen, then all others can not be chosen; this is fed back to the guessing part. In combination, since not all departments have to be excluded from selection, exactly one will be chosen in an answer set (provided some departments exist, otherwise $\emptyset$ is the single answer set). In other words, the only stable configurations of the above program are those in which one and only one atom of type $sel(v)$ is present.

Note that we could equally well replace the checking rule with the rule

$$-sel(D_1) \vee -sel(D_2) \leftarrow dept(D_1), dept(D_2), D_1 \neq D_2.$$

Informally, this rule says that if we have two different departments, then at least one of them can not be selected.

As a final example for choice, we consider a simple course scheduling scenario.

Example 39. Suppose there is a computer science department cs at a university u. We have information about members and courses of cs, as well as preferred courses of members, both encoded as facts F:

$$\begin{array}{r r r} member(sam, cs). & course(java, cs). & course(ai, cs). \\ member(bob, cs). & course(c, cs). & course(logic, cs). \\ member(tom, cs). & & \\ likes(sam, java). & likes(sam, c). & \\ likes(bob, java). & likes(bob, ai). & \\ likes(tom, ai). & likes(tom, logic). & \end{array}$$

Our task is now to assign each member of the department some courses, such that (i) each member should have at least one course, (ii) nobody should have more than two courses, and (iii) assign only courses that the course leader likes. We can use the following program P to encode this problem:

$$\begin{aligned}
teaches(X,Y) &\leftarrow member(X,cs), course(Y,cs), likes(X,Y),\\
&\quad \text{not } -teaches(X,Y).\\
-teaches(X,Y) &\leftarrow member(X,cs), course(Y,cs), teaches(X_1,Y), X_1 \neq X.\\
some\_course(X) &\leftarrow member(X,cs), teaches(X,Y).\\
&\leftarrow member(X,cs), \text{not } some\_course(X).\\
&\leftarrow teaches(X,Y_1), teaches(X,Y_2), teaches(X,Y_3),\\
&\quad Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3.
\end{aligned}$$

Informally, the first rule says that a CS faculty member gets a CS course she likes assigned by default. The second rule states that a CS faculty member does not get a CS course assigned if somebody else teaches it. The third and fourth rules make sure that each CS faculty gets at least one course assigned. The final rule excludes any assignment where one person is assigned three (or more) courses.

We obtain the following three answer sets of $P \cup F$:

– $\{teaches(sam,c), teaches(bob,java), teaches(bob,ai), teaches(tom,logic),\ldots\}$
– $\{teaches(sam,java), teaches(sam,c), teaches(bob,ai), teaches(tom,logic),\ldots\}$
– $\{teaches(sam,c), teaches(bob,java), teaches(tom,ai), teaches(tom,logic),\ldots\}$

6.3 Saturation Technique

A more advanced technique that is used in disjunctive ASP is the so called *saturation technique*, which is used to check whether *all* possible guesses satisfy a certain property, like *not* being a solution to a problem. Testing such a property, like whether all assignments of three colors to nodes do *not* legally color a graph G, may be co-NP-hard, and thus can not be evidently encoded in a normal logic program such that the program has some answer set precisely if G is *not* 3-colorable; in fact, the program in Example 33 has *no* answer set if G is *not* 3-colorable.

It is, however, possible to express the property of non-3-colorability by a *unique answer set candidate* for a program, such that the candidate is the only answer set if the graph is not 3-colorable, and is not an answer set otherwise. More abstractly, to test a property we design a program P and an answer set candidate M_{sat} such that M_{sat} is the single answer set of P if the property holds, and P has other answer sets (excluding M_{sat}) otherwise. The construction is such that any answer set of P is a subset of M_{sat}, and whenever the property is found to hold, any candidate answer set is "saturated" to M_{sat}. Intuitively, the property is tested *within* the answer set.

Example 40. For testing non-3-colorability, the constraints in the checking part of the program in Example 33 can be replaced, thus obtaining the program P_{non_col}:

$$\begin{aligned}
&b(X) \vee r(X) \vee g(X) \leftarrow node(X).\\
&non\_col \leftarrow r(X), r(Y), edge(X,Y).\\
&non\_col \leftarrow g(X), g(Y), edge(X,Y).\\
&non\_col \leftarrow b(X), b(Y), edge(X,Y).\\
&\chi(X) \leftarrow non\_col, node(X).
\end{aligned}$$

where $\chi \in \{r, g, b\}$. Informally, this change has the following effect: Whenever an assignment of colors to the nodes is bad, the assignment is rejected by "saturating" the candidate model at hand, selecting all ground facts $r(n)$, $g(n)$, and $b(n)$ for any node n. Importantly, the saturation is the same for all bad assignments. Thus, if *all* assignments of colors are bad, there will a be single answer set M_{sat} of the program, which contains, besides the graph description, non_col and $r(n)$, $g(n)$, and $b(n)$ for any node n. On the other hand, any good assignment of colors will lead to an answer set M such that $M \subset M_{sat}$, which means that M_{sat} is not an answer set of the program, and that P_{non_col} has many answer sets, smaller than M_{sat} corresponding to valid 3-colorings. Thus, M_{sat} is the single answer set of the program just if the graph is not 3-colorable. Note also that $P_{non\_col} \models_c non\_col$ iff the graph at hand is not 3-colorable.

We can abstract from the previous example a general design rule: if we desire to check that a property Pr holds *for all guesses* defining a search space, we can establish a guess and saturation check paradigm as follows:

- Define the search space of guesses through a subprogram P_{guess}, using disjunctive rules.
- Define a subprogram P_{check}, which checks Pr for a guess M_g.
- If Pr holds for M_g, an appropriate set of saturation rules P_{sat} generates the special candidate answer set M_{sat}.
- If Pr does not hold for M_g, an answer set results which is a *strict subset* of M_{sat} (thus preventing that M_{sat} is an answer set).

It is thus crucial that the program P_{check}, which formalizes Pr, and P_{sat} do not generate incomparable candidate answer sets. Incomparability might be easily introduced, besides subprograms with negation as failure, by improper use of disjunction, or by ill-designed (positive) saturation rules; we will see an example in the next subsection.

In combination with further guessing rules, which assign values to atoms that are not involved in saturation, it is possible to express problems that have complexity beyond NP, like the strategic companies problem [78,36], or quantified Boolean formulas of the form $\exists X \forall Y E(X, Y)$, which are Σ_2^p-complete.

6.4 Iteration over a Set

As last technique, we consider testing a property for all elements of a set without the use of negation. This may be needed in some contexts, for instance in combination with the saturation technique, or when the use of negation could lead to undesired behavior (e.g., in case of cyclic negation).

Example 41. Suppose we want to test whether in a directed graph $G = (V, E)$, all nodes are reachable from a designated start node $n_0 \in V$. Using the representation of G and n_0 as in Example 37, we could use the following rules and double negation:

$$\begin{aligned}
&all\_reached \leftarrow \text{not } -all\_reached.\\
&-all\_reached \leftarrow node(X), \text{not } reached(X).\\
&reached(X) \leftarrow start(X).\\
&reached(X) \leftarrow reached(Y),\ edge(Y, X).
\end{aligned}$$

Here, $all\_reached$ is true in the resulting answer set, exactly if all nodes are reachable from n_0.

Now suppose that we want to test in an answer set whether reachability holds for each graph G' that results from G by removing between all nodes n and n', that are mutually connected, one of the edges $n \rightarrow n'$, $n' \rightarrow n$ at random. The edges of G' can be generated using the rules

$$\begin{array}{l} edge_1(X,Y) \vee edge_1(Y,X) \leftarrow edge(X,Y), edge(Y,X). \\ edge_1(X,Y) \leftarrow edge(X,Y), \text{not}\, edge(Y,X). \end{array}$$

Let us replace $edge$ in the rules for $reached$ with $edge_1$ and add the saturation rule

$$edge_1(X,Y) \leftarrow all\_reached, edge(X,Y).$$

We thus obtain the program P_g:

$$\begin{array}{l} all\_reached \leftarrow \text{not } -all\_reached. \\ -all\_reached \leftarrow node(X), \text{not } reached(X). \\ reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ edge(Y,X). \\ edge_1(X,Y) \vee edge_1(Y,X) \leftarrow edge(X,Y), edge(Y,X). \\ edge_1(X,Y) \leftarrow edge(X,Y), \text{not}\, edge(Y,X). \\ edge_1(X,Y) \leftarrow all\_reached, edge(X,Y). \end{array}$$

However P_g does not work as expected, as evidenced, e.g., by the simple graph $G = (\{a,b\}, \{a \rightarrow b, b \rightarrow a\})$, where for $n_0 = a$ we get that the candidate

$$\begin{array}{rl} M_{sat} = \{ & all\_reached, edge_1(a,b), edge_1(b,a), reached(a), reached(b), \\ & edge(a,b), edge(b,a), node(a), node(b), start(a)\} \end{array}$$

is a "saturated" answer set, while the property fails for G (for a witnessing G', remove $a \rightarrow b$; we refer to this graph as $G_{-(a\rightarrow b)}$). Indeed, P has also an answer set $M_2 = \{-all\_reached, edge_1(b,a), reached(a), edge(a,b), \ldots\}$, which is not a subset of the saturation candidate, corresponding to the graph $G_{-(a\rightarrow b)}$.

This apparently non-obvious behaviour can be explained from several perspectives: M_{sat} is a proper answer set due to the fact that $G_{-(b\rightarrow a)}$ reaches all possible nodes from a, thus making $all\_reached$ true. Consequently, the saturation rule makes the extension of $edge_1$ equal to $edge$, which results in M_{sat}. On the other hand, one might expect that M_2, which corresponds to the deletion of the edge $a \rightarrow b$, should invalidate M_{sat}, and thus obtain M_2 as (the single) answer set. But M_2 *is not* contained in M_{sat}. Indeed, although M_2 is not a saturated answer set, and although the extension of $edge_1$ in M_2 is a strict subset of $edge$, it contains the "spare" atom $-all\_reached$, which does not appear in M_{sat}, making the two answer sets incomparable.

We are thus in a situation in which M_{sat} is an answer set, resulting from *some* of the guessed subgraphs G' of G in which reachability is retained, while we expected M_{sat} as an answer set if and only if reachability holds *for all* guessed subgraphs G' of G.

The problems of program P_g can be remedied by using recursive positive rules—which check whether each node is reachable—instead of double negation. This will help in establishing a "proper" containment between candidate answer sets and M_{sat}. To this

$\%$ Guess a subgraph for testing
$edge_1(X,Y) \vee edge_1(Y,X) \leftarrow edge(X,Y), edge(Y,X).$
$edge_1(X,Y) \leftarrow edge(X,Y), \text{not } edge(Y,X).$
$\%$ Compute all reachable nodes
$reached(X) \leftarrow start(X).$
$reached(X) \leftarrow reached(Y),\ edge_1(Y,X).$
$\%$ iterate to check if all nodes are reached
$all\_reached \leftarrow last(X), all\_reached\_upto(X).$
$all\_reached\_upto(X) \leftarrow all\_reached\_upto(Y), succ(Y,X), reached(X).$
$all\_reached\_upto(X) \leftarrow first(X), reached(X).$
$\%$ Saturation rule
$edge_1(X,Y) \leftarrow all\_reached, edge(X,Y).$

Fig. 8. Program with reachability test for subgraphs in an answer set

end, an ordering of the nodes is taken, and associated successor predicates $first(X)$, $succ(Y,X)$, and $last(X)$ which express that X is the first node, the successor of Y, and the last node in this ordering, respectively. The rules for $all\_reached$ and $-all\_reached$ in P are replaced by the following rules:

$$all\_reached \leftarrow last(X), all\_reached\_upto(X).$$
$$all\_reached\_upto(X) \leftarrow all\_reached\_upto(Y), succ(Y,X), reached(X).$$
$$all\_reached\_upto(X) \leftarrow first(X), reached(X).$$

if we add then the respective facts for the ordering, the resulting program (see Figure 8) works as desired. Informally, these rules access the (positive) reachability that is computed by the rules for $reached$ with respect to varying subgraphs G' of G.

The use of an ordering and a rule scheme as above can be easily applied in other contexts. In case the set over which the iteration is made is susceptible to change itself (e.g., if in the previous example only all nodes that have no outgoing edges need to be reached), then special rules can be added that skip elements, indicated by $out(X)$:

$$all\_reached\_upto(X) \leftarrow all\_reached\_upto(Y), succ(Y,X), out(X).$$
$$all\_reached\_upto(X) \leftarrow first(X), out(X).$$

where out is computed using positive rules; the formulation for the continued example is left as a (simple) exercise.

7 Answer Set Solvers

In this section, we mention some AS solvers and briefly present the DLV system.

Given that deciding whether a given extended logic program has some answer set is NP-complete, it is clear that efficient computation of answer sets is not easy, and that we can not expect to have a polynomial time algorithm for this task (even under polynomial total-time, i.e., if the combined size of the input P and the output in terms of all answer sets of P is measured). In fact, the problem is yet harder if disjunction in rule heads is allowed.

Table 1. Some Answer Set Solvers

DLV	http://www.dbai.tuwien.ac.at/proj/dlv/ [a]
Smodels	http://www.tcs.hut.fi/Software/smodels/ [b]
GnT	http://www.tcs.hut.fi/Software/gnt/
Cmodels	http://www.cs.utexas.edu/users/tag/cmodels/
ASSAT	http://assat.cs.ust.hk/
NoMore(++)	http://www.cs.uni-potsdam.de/~linke/nomore/
Platypus	http://www.cs.uni-potsdam.de/platypus/
clasp	http://www.cs.uni-potsdam.de/clasp/
XASP	http://xsb.sourceforge.net, distributed with XSB v2.6
aspps	http://www.cs.engr.uky.edu/ai/aspps/
ccalc	http://www.cs.utexas.edu/users/tag/cc/

a + several extensions, e.g. dlvhex, dlv-db, dlt, OntoDLV
b + Smodels_cc

A number of different, sophisticated algorithms have been developed over the past 15 years (similar as in the area of SAT solving), and to date a number of AS solvers are available; a partial list is shown in Table 1. Some of these solvers provide a number of extensions to the language described here, and have been further developed into families of solvers (e.g. the DLV system).

A collection of benchmark problems for AS solvers is maintained at the ASPARAGUS platform,[13] where also information about language formats and the ASP System Competition (whose first edition was at the LPNMR 2007 conference) can be found. In the next subsection, we briefly address implementation strategies of AS solvers; an excellent source on this topic is Ilkka Niemelä's ICLP'04 tutorial.[14]

7.1 Architecture of ASP Solvers

Traditional answer set solvers typically have a two level architecture:

1. **Grounding Step:** Given a program P with variables, a (subset) P' of its grounding is generated which has the same answer sets as P.
2. **Model Search**: The answer sets of the grounded (propositional) program P' are computed.

Thus, in analogy with the definition of the semantics, also the computation proceeds by a reduction to the propositional case. To facilitate finite computations (and answer sets), many systems do not support function symbols (that is, they handle the so called *Datalog* fragment of logic programming), or only in a very limited form; this is because as already mentioned, function symbols are a well-known source of undecidability, even in rather plain settings (see [25]); see Section 9 for further discussion.

The efficient realization of both steps requires the use of sophisticated algorithms and methods; some have been developed from scratch, while others have been borrowed from related areas, e.g., from SAT Solving. We next look at the two steps.

[13] http://asparagus.cs.uni-potsdam.de/
[14] http://www.tcs.hut.fi/~ini/papers/niemela-iclp04-tutorial.ps.gz.

Grounding Step. Efficient grounding is at the heart of current state-of-the art systems, and different grounding procedures have been realized, including

- DLV's grounder (integrated),
- lparse (for Smodels), gringo (for clasp), which can be used separately, and
- XASP, aspps.

In order to make the ground program P' small and easy to evaluate, sophisticated techniques for "intelligent grounding" have been developed, and restrictions are imposed on the rule syntax, like

- *rule safety* (DLV): every variable in a rule must occur in some positive body literal (i.e., not prefixed with not) whose predicate is not '$=$" or any another built-in comparison predicate. This is a standard condition in the area of deductive databases.
- *domain-restriction* (Smodels): every variable in a rule must occur in a positive *domain predicate*, which are predicates not defined via negative recursion or using "choice rules" [120].

A problem with even highly efficient grounding procedures is that in the end a *grounding bottleneck* may show up: even if a given program P can be evaluated in polynomial space in principle, the (small) grounding P' produced might contain exponentially many rules; this is discussed in detail in [35]. Efficient nonground evaluation of ASP programs intensified only more recently. Techniques for partial and lazy grounding (as used e.g. in [20,58,101,76]) are proving to be helpful and thus naturally constitute an important issue for the next generation of AS solvers.

Model search. The second step is model (answer set) search for a propositional program. This is more complicated than the analog problem in SAT Solving or CSP, as it informally comprises two subtasks:

- generation of a candidate model (e.g., a classical model), and
- model checking (testing the stability condition); this problem is easily shown to be P-complete for normal programs and to be co-NP-complete for disjunctive programs, respectively.

The two tasks can be solved using different approaches:

1. One approach, which is historically the first, employs special model search algorithms. Such algorithms have been developed, e.g., for Smodels, DLV, NoMore, aspps, and clasp. They take inspiration from the DPLL algorithm for SAT and its variants and improvements, in which truth values are assigned to atoms, consequences that emerge propagated and, if conflicts are found, backtracking takes place. However, while a SAT solver may find any classical model, an AS solver has to find a *specific* such model which satisfies stability; this makes the task much harder. E.g., only atoms can be true that are supported by rules. The result of the model search is an answer set candidate, whose stability may still need to be checked, as it is the case for disjunctive programs in DLV, for instance. To this

a:– not b.
b:– not a.
c:– not c, a.

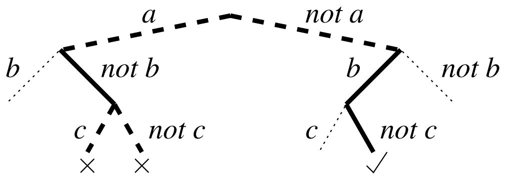


Fig. 9. Model search for a simple program (solid lines = deterministic propagation)

end, the characterization of stable models in terms of unfounded sets according to Theorem 11 is exploited (which can be compiled into an instance of UNSAT).

The search for a specific model led in the DLV system, e.g. to four truth values for an atom in the search: *t(rue), f(alse), u(ndefined)*, and *m(ust-be-true)*. Here *must-be-true* means that the atoms has to be true, but its truth still remains to be supported. Starting with all atoms undefined, *possibly true* literals are identified, whose truth value is subsequently determined with trial and error. For a simple example, consider the program in Figure 9. Initially, all atoms are undefined and not a and not b would be possibly true literals. Assume that first a is assigned false; then the right branch is explored. For b, we then can conclude from the first rule must-be-true (as its body must be false) and true from the second rule (as its body is false); hence, b is assigned true. For c, we can conclude false (as a is false). Now all atoms are either true or false; the candidate $M = \{b\}$ is indeed an answer set and output. Coming back to the root, a is alternatively assigned must-be-true. From the first rule, we would then conclude that b is false, which in turn makes a true; further, one would conclude that not c is possibly true. However, setting c to false leads to a conflict (by the third rule c then would have to be true), and also setting c to must-be-true, as then c has no support and must be false.

Important for these approaches are heuristics (which atom/rule to consider next); for more details concerning DLV, see [52,90].

2. Later, another approach was to translate the logic program to SAT Solving, which has been realized e.g. in ASSAT and Cmodels. To this end, as already mentioned in Section 5.3 the so called *Clark completion* of a logic program [23] (which translates an acyclic program into an equivalent SAT instance) is extended with loop formulas [87,75].

   Note that for SAT, model checking is easy in terms of complexity, and can be done in LOGSPACE (in fact, the problem is solvable in ALOGTIME, which is far down in LOGSPACE).[15] An attractive advantage of this approach is that it can benefit from improvements to SAT solving technology; drawbacks are that to generate all answer sets, one needs a SAT solver that can compute all models of a clause set (or one has to tune the transformation for incremental enumeration of answer sets) and that in general, the SAT instance that is constructed can have exponential size.

[15] This also intuitively is a clue why there are so many loop formulas, made more precise in [85]: if there were few and they could be easily constructed, we could solve a P-complete (resp. co-NP-complete) problem in LOGSPACE (resp., in polynomial time).

7.2 The DLV System

As an example of an AS solver, we briefly consider here the DLV system.[16] DLV is a state-of-the-art answer set solver which has been developed at the Vienna University of Technology and the University of Calabria over more than a decade, starting out with a research project on non-monotonic deductive databases in 1996; it is freely available for download.

The system has a language that is richer than the extended disjunctive logic programs considered above, and supports additional constructs (e.g., aggregates, weak constraints) some of which increase the expressivity (e.g., weak constraints allow to express optimization problems with complexity beyond Σ_2^p). DLV supports certain built-in predicates (e.g. bounded integer arithmetic and comparisons), and offers a range of front-ends for specific KR tasks (e.g., planning or diagnosis), as well an interface to databases. The principle reasoning tasks supported by DLV are 1. answer set generation (all or a given number) and 2. brave and cautious query answering, which is supported for both ground and non-ground queries.

The DLV system has been described in many publications, of which [78] is the most comprehensive; the article [77] is recent. As mentioned above, its reasoning engine implements the two-level architecture outlined in Section 7, using a highly optimized grounding module for the first level; the model search is by a DPLL style algorithm that uses the characterization of stable models by unfounded sets in Theorem 11. DLV also incorporates a lot of deductive database technology in order to handle larger data volumes (including magic sets, which have been generalized to programs with negation and disjunction). The DLV engine has been extended in many directions leading to a family of systems that support different purposes, including dlv-ex, dlvhex, OntoDLV, dlv-db, and dlt.

DLV syntax. Briefly, the core language of DLV consists of rules of the form

$$\mathtt{a_1\ v\ \cdots\ v\ a_n\ :-\ b_1, \cdots, b_k,\ not\ b_{k+1}, \cdots,\ not\ b_m.}$$

where $n + m > 1$, and all $\mathtt{a_i}$, $\mathtt{b_j}$ are atoms or strongly negated atoms (e.g. $-\mathtt{a}$); no function symbols are allowed; the syntax of terms is like in Prolog. Certain built-in predicates are supported (cf. Table 2). Note that DLV allows constraints ($n = 0$); as mentioned in Section 7.1, DLV requires rule safety. The extended language also allows that the $\mathtt{b_j}$ are aggregate atoms, in which the values of aggregate functions over a conjunction of literals (including $\#\max$, $\#\min$, #sum, #count, and #times), can be compared to given bounds; we refer to [53] for details.

Furthermore, the DLV language has *weak constraints*, which are of the form

$$:\sim \mathtt{b_1, \cdots, b_k,\ not\ b_{k+1}, \cdots,\ not\ b_m.}\ [w : l] \tag{28}$$

where all $\mathtt{b_j}$ are as in rules and w (the *weight*) and l (the *level*, or *layer*) are positive integer constants or variables. For convenience, w and/or l can be omitted and are set to 1 in this case. Informally, the expression (28) is a constraint that can be violated, which incurs cost w; for an answer set, the costs of all violated (instances of) weak constraints

[16] http://www.dbai.tuwien.ac.at/proj/dlv/

Table 2. DLV built-ins (Oct-11-2007 release)

Comparison Predicates (for constants and integers):

$<, >, <=, >=, ==, ! =$

Arithmetic Predicates (require an upper bound #`maxint` for integers; see below):

$\#\mathtt{int}(\mathtt{X})$:	$\mathtt{X}$ is known integer $(\mathtt{1} \leq \mathtt{X} \leq \mathtt{N})$.
$\#\mathtt{succ}(\mathtt{X},\mathtt{Y})$:	$\mathtt{Y}$ is successor of $\mathtt{X}$, i.e., $\mathtt{Y} = \mathtt{X} + \mathtt{1}$.
$+(\mathtt{X},\mathtt{Y},\mathtt{Z})$:	$\mathtt{Z} = \mathtt{X} + \mathtt{Y}$.
$*(\mathtt{X},\mathtt{Y},\mathtt{Z})$:	$\mathtt{Z} = \mathtt{X} * \mathtt{Y}$.

Facts over a fixed integer range

$\mathtt{pred}(c_1..c_2)$.	where $c_1, c_2 \geq 0$ are numeric integer constants, is short for $\mathtt{pred}(c_1)$. $\mathtt{pred}(c_1{+}1)$. $\cdots \mathtt{pred}(c_2 - 1)$. $\mathtt{pred}(c_2)$.

Built-in constants

#`maxint`	upper integer limit, set with -N switch, or with #`maxint` $= i$. for integer $i \geq 0$ in the program

are added up, grouped by levels of priorities l. Among all answer sets, those whose cost vector is lexicographically (ordered by priority) smallest are chosen as *optimal answer sets*; see [78] for formal details. With the help of weak constraints, the Guess and Check methodology in Section 6.2 can be extended to a Guess, Check and Optimize Methodology (an example follows shortly).

Queries are specified in DLV by expressions

$$\mathtt{b_1}, \cdots, \mathtt{b_k},\ \mathtt{not}\ \mathtt{b_{k+1}}, \cdots,\ \mathtt{not}\ \mathtt{b_m}?$$

where all $\mathtt{b_j}$ are atoms or strongly negated atoms; the query mode (brave or cautious) is selected using a switch (–brave resp. –cautious). Variables in queries are allowed; all bindings of variables to constants will be shown such that the resulting ground query evaluates to true; if the query is ground, in case of brave reasoning a witnessing answer set is shown.[17]

Example 42. To illustrate the use of weak constraints, we consider the well-known Traveling Salesperson problem (TSP), where cities are stored as facts $city(X)$ and direct connections as facts $conn(X, Y, C)$, where C is the cost of traveling from X to Y. Furthermore, the tour is required to start in a city designated with a fact $start(X)$.

The following DLV program computes optimal tours in its optimal answer sets:

```
inTour(X, Y, C) v outTour(X, Y, C) :- start(X), conn(X, Y, C).     ⎫
inTour(X, Y, C) v outTour(X, Y, C) :- reached(X), conn(X, Y, C).   ⎬ Guess
reached(X) :- inTour(Y, X, C).                               (aux.)⎭
```

[17] Unless magic sets are enabled, which will be the default in future DLV releases.

$$\left.\begin{array}{l}
:\!-\ \mathtt{inTour(X,Y,\_), inTour(X,Y1,\_), Y <> Y1.}\\
:\!-\ \mathtt{inTour(X,Y,\_), inTour(X1,Y,\_), X <> X1.}\\
:\!-\ \mathtt{city(X), not\ reached(X).}
\end{array}\right\}\ \textbf{Check}$$

$$:\!\sim\ \mathtt{inTour(X,Y,C).\ [C:1]}\quad\Big\}\ \textbf{Optimize}$$

Here, the first three rules guess a tour, which is done in an incremental manner beginning at the start city for all cities reached. Note that different from Example 37, we want a complete tour (a cycle) rather than a path and thus the start city must be reached from some other city. The weak constraint in the optimize part states that including the connection from X to Y costs C; the total cost of an answer set is the cost of a tour (all weak constraints have the same priority).

If we add the query

```
start(X), inTour(X, Y, C)?
```

to the program, then we obtain under brave reasoning all possible first legs for an optimal tour, and under cautious reasoning a mandatory first leg (if one exists).

Front-ends. Besides the answer set semantics core, DLV offers various front-ends for particular KR tasks, including

– diagnosis
– knowledge-based planning ($\mathcal{K}$ language)
– front-end to SQL3
– inheritance reasoning

The first three front-ends can be invoked using command-line switches, while the inheritance front-end is automatically enabled if the input is an inheritance program [17]. Many other front-ends, created by various authors, are available as separate packages.

Using DLV. The DLV system is primarily command-line oriented, but there is also a plain GUI and there are web interfaces available.[18]

The system reads input from files whose names are passed on the command-line. If the command-line option "--" has been specified, input is also read from standard input (stdin). Output is printed to standard output (stdout), one line per answer set. Detailed documentation and an online manual are available at the DLV homepage. [16]

8 ASP for the Semantic Web

As mentioned in Section 6, ASP has been deployed to many application areas. We focus here on the Semantic Web, where different ways to exploit ASP and ASP techniques have been considered (see [44,41,34] for more discussion):

– As a host language for Web/Semantic Web formalisms. For example, mappings respectively encodings of ontologies in description logics into ASP have been conceived (see [43] for references), and encoding of web query languages, e.g. SPARQL [105].

[18] E.g. http://asptut.gibbi.com/

- For diverse problem solving, like Web service composition (e.g. [110,107]), Web Service repair [57], or ontology merging [47,69].
- For combining rules and ontologies into a unifying framework (cf. [43,27,33] for discussion and references).

In the context of the Semantic Web, special needs arise that have to be accommodated:

- dealing with open worlds and domains (cf. [106,24]),
- access to (semi-)structured and poorly structured data,
- external sources and distributed computation (cf. [2]),
- heterogeneity of sources, and
- web dynamics (cf. [1]).

In the rest of this section, we review some research and development efforts which have been moving ASP languages in the direction of the Semantic Web. Among them are extensions of ASP to access ontologies in OWL, the Web Ontology Language, and extensions which allow to access heterogeneous knowledge sources on the Web. For more information on ASP for the Semantic Web, we refer to previous Reasoning Web schools [41,44] and the tutorial [34].

8.1 DL-Programs

Description logic programs (dl-programs), which had been introduced in [49], are a form of hybrid knowledge bases combining description logics[19] and logic programs under answer set semantics. They form another contribution to the attempt in finding an appropriate formalisms for combined rules and ontologies for the Semantic Web.

Roughly speaking, dl-programs consist of a normal logic program P and a description logic knowledge base (DL-KB) L. In addition to traditional atoms, the logic program P might contain special devices, called dl-atoms. Those dl-atoms may occur in the body of a rule and involve queries to L. Moreover, dl-atoms can specify an input to L before querying it, thus in dl-programs a bidirectional data flow is possible between the description logic component and the logic program.

The way dl-programs interface DL-KBs enables the possibility of acting as a loosely coupled formalism between a knowledge base formulated in terms of a logic program and a knowledge base formulated in terms of description logic axioms. This feature brings the advantage of reusing existing logic programming and DL systems in order to build an implementation of dl-programs.

In the following, we provide the syntax of dl-programs and an overview of the semantics. An in-detail treatise is given in [43].

We will illustrate the main ideas behind the notion of dl-program with the following example:

Example 43. An existing network must be extended by new nodes (Fig. 10). The knowledge base L_N contains information about existing nodes $(n_1, \ldots, n_5)$ and their

[19] The reader is referred to [8] of this volume for a general background on description logics.

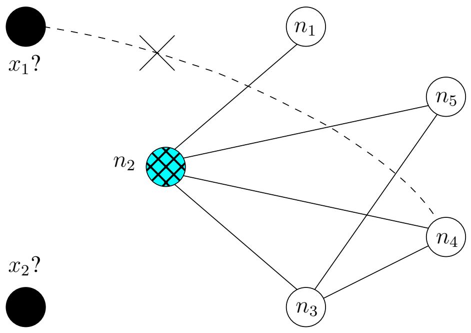


Fig. 10. Hightraffic network

interconnections as well as a definition of "overloaded" nodes (concept *HighTrafficNode*), which are nodes with more than three connections:

$$\begin{aligned}
&\geq 1\ \mathit{wired} \sqsubseteq \mathit{Node};\ \ \top \sqsubseteq \forall \mathit{wired}.\mathit{Node};\ \ \mathit{wired} = \mathit{wired}^{-};\\
&\geq 4\ \mathit{wired} \sqsubseteq \mathit{HighTrafficNode};\ \ n_1 \neq n_2 \neq n_3 \neq n_4 \neq n_5;\\
&\mathit{Node}(n_1);\ \ \mathit{Node}(n_2);\ \ \mathit{Node}(n_3);\ \ \mathit{Node}(n_4);\ \ \mathit{Node}(n_5);\\
&\qquad\qquad \mathit{wired}(n_1, n_2);\ \ \mathit{wired}(n_2, n_3);\ \ \mathit{wired}(n_2, n_4);\\
&\qquad\qquad \mathit{wired}(n_2, n_5);\ \ \mathit{wired}(n_3, n_4);\ \ \mathit{wired}(n_3, n_5).
\end{aligned}$$

In L_N, only n_2 is an overloaded node, and is highlighted in Fig. 10 with a criss-cross pattern.

To evaluate possible combinations of connecting the new nodes, the following program P_N is specified:

$$\mathit{newnode}(x_1). \tag{29}$$

$$\mathit{newnode}(x_2). \tag{30}$$

$$\mathit{overloaded}(X) \leftarrow \mathrm{DL}[\mathit{wired} \uplus \mathit{connect}; \mathit{HighTrafficNode}](X). \tag{31}$$

$$\begin{aligned}\mathit{connect}(X, Y) \leftarrow\ & \mathit{newnode}(X), \mathrm{DL}[\mathit{Node}](Y),\\ & \text{not}\ \mathit{overloaded}(Y), \text{not}\ \mathit{excl}(X, Y).\end{aligned} \tag{32}$$

$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(X, Z), \mathrm{DL}[\mathit{Node}](Y), Y \neq Z. \tag{33}$$

$$\mathit{excl}(X, Y) \leftarrow \mathit{connect}(Z, Y), \mathit{newnode}(Z), \mathit{newnode}(X), Z \neq X. \tag{34}$$

$$\mathit{excl}(x_1, n_4). \tag{35}$$

Rules (29)–(30) define the new nodes to be added. Rule (31) imports knowledge about overloaded nodes in the existing network, taking new connections already into account. Rule (32) connects a new node to an existing one, provided the latter is not overloaded and the connection is not to be disallowed, which is specified by Rule (33) (there must not be more than one connection for each new node) and Rule (34) (two new nodes cannot be connected to the same existing one). Rule (35) states a specific condition: Node x_1 must not be connected with n_4.

Two different semantics have been defined for dl-programs, the (strong) answer-set semantics [49] and the well-founded semantics [50,42]. The former extends the notion of Gelfond-Lifschitz reduct (see Section 4) incorporating the presence of dl-atoms: dl-programs can have, in general, multiple answer sets. The latter extends the well-founded semantics of [122] to dl-programs.

Example 44. As specified by the strong answer set semantics of dl-programs, the program (L_N, P_N) in Example 43 has four strong answer sets (we show only atoms with predicate *connect*): $M_1 = \{\mathit{connect}(x_1, n_1),\ \mathit{connect}(x_2, n_4),\ \ldots\}$, $M_2 = \{\mathit{connect}(x_1, n_1),\ \mathit{connect}(x_2, n_5), \ldots\}$, $M_3 = \{\mathit{connect}(x_1, n_5),\ \mathit{connect}(x_2, n_1), \ldots\}$, and $M_4 = \{\mathit{connect}(x_1, n_5),\ \mathit{connect}(x_2, n_4),\ \ldots\}$. Note that the ground dl-atom

$$\mathrm{DL}[\mathit{wired} \uplus \mathit{connect}; \mathit{HighTrafficNode}](n_2)$$

from rule (31) is true in any partial interpretation of P_N. According to the proposed well-founded semantics for dl-programs in [50], the unique well-founded model of (L_N, P_N) contains thus $\mathit{overloaded}(n_2)$.

Features and Properties of DL-Programs. The strong answer set semantics of dl-programs is nonmonotonic, and generalizes the stable semantics of ordinary logic programs. In particular, satisfiable positive dl-programs (programs without default negation and $\cap\!\!\!-$ operator) have a least model semantics, and satisfiable stratified dl-programs have a unique minimal model which is iteratively described by a finite sequence of least models.

Applications. The bidirectional flow of knowledge between a description logic base and a logic program component enables a variety of possibilities. A major application for dl-programs is nonmonotonic reasoning on top of monotonic systems. It is for instance possible to take a dl-knowledge base L and coupling it with a properly designed logic program in order to extend L with *defaults* [112] and *closed world assumption* (CWA) [111]. Both reasoning applications can be implemented in dl-programs to support nonmonotonic reasoning for description logics, as detailed in [43].

8.2 HEX-Programs

HEX-programs [46] are declarative nonmonotonic logic programs with support for external knowledge and higher-order disjunctive rules, under answer set semantics. In spirit of dl-programs, they allow for a loose coupling between general external knowledge sources and declarative logic programs through the notion of external atoms, which take input from the logic program and exchange inferences with the external source. In addition, meta-reasoning tasks may be accomplished by means of higher-order atoms. HEX-programs are evaluated under a generalized answer-set semantics, thus are in principle capable of capturing many proposed extensions in answer-set programming.

Syntax of HEX-Programs. Let $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, and *external predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case), while elements from $\mathcal{G}$ are prefixed with the "&" symbol. We note that constant names serve both as individual and predicate names.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms; $n \geq 0$ is the *arity* of the atom. Intuitively, Y_0 is the predicate name, and we thus also use the more familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant.

For example, $(x, \mathit{rdf}{:}\mathit{type}, c)$, $\mathit{node}(X)$, and $D(a, b)$, are atoms; the first two are ordinary atoms.

An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m) \ , \tag{36}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&g \in \mathcal{G}$ is an external predicate name. We assume that $\&g$ has fixed lengths $\mathit{in}(\&g) = n$ and $\mathit{out}(\&g) = m$ for input and output lists, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates: in this respect, an external predicate $\&g$ is equipped with a function $f_{\&g}$ evaluating to true for proper input values.

A *rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \mathrm{not}\, \beta_{m+1}, \ldots, \mathrm{not}\, \beta_n \ , \tag{37}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_n$ are either atoms or external atoms. We define $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ and $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*; r is *ordinary*, if it contains only ordinary atoms. A HEX-*program* is a finite set P of rules. It is *ordinary*, if all rules are ordinary.

We next give an illustrative example.

Example 45 ([45]). Consider the HEX-program P in Figure 11. Informally, this program randomly selects a certain number of John's relatives for invitation. The first line states that $\mathit{brotherOf}$ is a subrelation of $\mathit{relativeOf}$, and the next three lines give concrete facts. The disjunctive rule (42) chooses relatives, employing the external predicate $\&\mathit{reach}$. This latter predicate takes in input a binary relation e and a node name n, returning the nodes reachable from n when traversing the graph described by e (see the following Example 47). Rule (43) axiomatizes subrelation inclusion exploiting higher-order atoms; that is, for those couples of binary predicates p, r for which it holds $\mathit{subRelation}(p, r)$, it must be that $r(x, y)$ holds whenever $p(x, y)$ is true.

The constraints (45) and (46) ensure that the number of invitees is between 1 and 2, using (for illustration) an external predicate $\&\mathit{degs}$ from a graph library. Such a predicate has a valuation function $f_{\&\mathit{degs}}$ where $f_{\&\mathit{degs}}(I, e, \mathit{min}, \mathit{max})$ is true iff min and max are, respectively, the minimum and maximum vertex degree of the graph induced by the edges contained in the extension of predicate e in interpretation I.

$$subRelation(brotherOf, relativeOf). \quad (38)$$
$$brotherOf(john, al). \quad (39)$$
$$relativeOf(john, joe). \quad (40)$$
$$brotherOf(al, mick). \quad (41)$$
$$invites(john, X) \vee skip(X) \leftarrow X \neq john, \&reach[relativeOf, john](X). \quad (42)$$
$$R(X, Y) \leftarrow subRelation(P, R), P(X, Y). \quad (43)$$
$$someInvited \leftarrow invites(john, X). \quad (44)$$
$$\leftarrow \text{not } someInvited. \quad (45)$$
$$\leftarrow \&degs[invites](Min, Max), Max > 2. \quad (46)$$

Fig. 11. Example HEX program

Semantics of HEX-Programs. In the sequel, let P be a HEX-program. The *Herbrand base* of P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule r, $grnd(r)$, is defined accordingly, and the grounding of program P is given by $grnd(P) = \bigcup_{r \in P} grnd(r)$. Unless specified otherwise, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly given by P.

Example 46 ([45]). Given $\mathcal{C} = \{edge, arc, a, b\}$, ground instances of $E(X, b)$ are for instance $edge(a, b)$, $arc(a, b)$, $a(edge, b)$, and $arc(arc, b)$. Ground instances of $\&reach[edge, N](X)$ are all possible combinations where N and X are replaced by elements from $\mathcal{C}$; some examples are $\&reach[edge, edge](a)$, $\&reach[edge, arc](b)$, and $\&reach[edge, edge](edge)$.

An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms. We say that I is a *model* of atom $a \in HB_P$, denoted $I \models a$, if $a \in I$.

With every external predicate name $\&g \in \mathcal{G}$, we associate an $(n+m+1)$-ary Boolean function $f_{\&g}$ assigning each tuple $(I, y_1 \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where $n = in(\&g)$, $m = out(\&g)$, $I \subseteq HB_P$, and $x_i, y_j \in \mathcal{C}$. We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, if and only if $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$.

Example 47 ([45]). Let us associate with the external atom $\&reach$ a function $f_{\&reach}$ such that $f_{\&reach}(I, E, A, B) = 1$ iff B is reachable in the graph E from A. Let $I = \{e(b, c), e(c, d)\}$. Then, I is a model of $\&reach[e, b](d)$ since $f_{\&reach}(I, e, b, d) = 1$.

Note that in contrast to the semantics of higher-order atoms, which in essence reduces to first-order logic as customary (cf. [115]), the semantics of external atoms is in spirit of second order logic since it involves predicate extensions.

Considering example 45, as John's relatives are determined to be Al, Joe, and Mick, P has six answer sets, each of which contains one or two of the facts $invites(john, al)$, $invites(john, joe)$, and $invites(john, mick)$.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and

(iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in grnd(P)$. We call P *satisfiable*, if it has some model.

Given a HEX-program P, the *FLP-reduct* of P with respect to $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in grnd(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of fP^I.

In principle, the truth value of an external atom depends on its input and output lists and on the entire model of the program. In practice, however, we can identify certain types of input terms that allow to restrict the input interpretation to specific relations. The Boolean function associated with the external atom $\&reach[edge, a](X)$ for instance will only consider the extension of the predicate $edge$ and the constant value a for computing its result, and simply ignore everything else of the given input interpretation.

Features and Properties of HEX-Programs. As mentioned above, HEX-programs are a generalization of dl-programs, consisting indeed in a form of coupling of rules with arbitrary external computation sources, within a declarative logic-based setting. The higher-order features are similar to those of HiLog [22], i.e., the semantics of this high-order extension is still within first-order logic.

The semantics of HEX-programs conservatively extends ordinary answer-set programs, and it is easily extendable to support weak constraints [18]. External predicates can define other ASP features like aggregate functions [53]. Computational complexity of the language depends on external functions. The former is however not affected if external functions evaluate in polynomial time.

The dlvhex prototype,[20] an implementation of HEX-programs, is based on a flexible and modular architecture. The evaluation of the external atoms is realized by plugins, which are loaded at run-time. The pool of available external predicates can be easily customized by third-party developers.

Applications. HEX-programs have been put to use in many applications in different contexts. Hoehndorf et al. [69] showed how to combine multiple biomedical upper ontologies by extending the first-order semantics of terminological knowledge with default logic. The corresponding prototype implementation of such kind of system is given by mapping the default rules to HEX-program. Fuzzy extensions of answer-set programs and their relationship to HEX-programs are given in [98,68]. The former maps fuzzy answer set programs to HEX-programs, whereas the latter defines a fuzzy semantics for HEX-programs and gives a translation to standard HEX-programs. In [99], the planning language $\mathcal{K}^c$ has been introduced which features external function calls in spirit of HEX-programs.

8.3 Other Linguistic Extension of ASP in the Direction of Semantic Web

We briefly survey here other notable works aiming at integrating the stable model semantics with Semantic Web related formalisms, and remind the reader to other discussions of related work such as [41,27].

[20] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

Research efforts can be categorized in the two main groups of *translational approaches* and *integration approaches*. The latter can be further classified in *loose, tight* or *full* integration.

As for integration approaches, $\mathcal{DL}+log$ [114] is the latest in a chain of extensions of the DL $\mathcal{ALC}$ with rules such as $\mathcal{AL}$-*log*, r- and r$^+$-hybrid knowledge bases. As a tight semantics approach, $\mathcal{DL}+log$ gives meaning to combined knowledge bases in terms of unique model structures, which aim at satisfying both the description logic base at hand and the logic program. *Hybrid MKNF knowledge bases* [95] build on Lifschitz's bimodal *Logic of Minimal Knowledge and Negation as Failure (MKNF)* [80], and aim at a seamless (which is sometimes referred as *full* integration) integration of classic and nonmonotonic semantics beyond tight integration approaches. Besides dl-programs and hex-programs, it is worth mentioning other loose coupling languages in the direction of probabilistic [88] and fuzzy hybrid systems [89] under stable semantics; see [33] for an overview. An extension of RDF(S) with stable models has been proposed in [3].

One might also consider the idea of translating Semantic Web ontologies to semantically equivalent ASP logic programs. This task is quite challenging, given the profound semantic differences between the two formalisms. Nonetheless, some success has been reached, and translation from several flavors of description logics to ASP are known, cf. [10,118,70,116]. Notably, the availability of function symbols (or, in any case, of infinite domains), solves some of the semantic difficulties [116,67].

8.4 Other Semantic Web Enabled Systems Based on ASP

OntoDLV [113] is a system for ontologies specification and reasoning under answer set semantics. OntoDLV implements a logic-based ontology representation language, called OntoDLP (where DLP stands for Disjunctive Logic Programs), which is an extension of (disjunctive) ASP with all the main ontology constructs including classes, inheritance, relations, and axioms. OntoDLP is strongly typed, and includes also complex type constructors, like lists and sets. OntoDLV supports some interoperability mechanism with OWL, allowing the user to retrieve information from external OWL Ontologies and to exploit this data in OntoDLP ontologies and queries. OntoDLV facilitates the development of complex applications in a user-friendly visual environment; it is endowed with a robust persistency-layer for saving information transparently on a DBMS, and it seamlessly integrates the `DLV` system [78] exploiting the power of a stable and efficient AS solver. Indeed, OntoDLV is already used for the development of real-world applications including agent-based systems, information extraction and text classification frameworks.

GiaBATA [71] is a system for storing, aggregating, and querying Semantic Web data, based on declarative logic programming technology, namely on the dlvhex system, which allows to implement a fully SPARQL compliant semantics, and on `dlv-db`, which extends the `DLV` system with persistent storage capabilities. Compared with off-the-shelf RDF stores and SPARQL engines[21], GiaBATA offers more flexible support for rule-based RDFS and other higher entailment regimes by enabling custom reasoning via

[21] For details of RDF and its query language SPARQL, the reader may refer to [6] in this volume.

rules, and the possibility to choose the reference ontology on a per query basis. Due to the declarative approach, GiaBATA gains the possibility of applying well-known logic-level optimization features of logic programming (LP) and deductive database systems. The architecture of GiaBATA allows for extensions of SPARQL by non-standard features such as aggregates, custom built-ins, or arbitrary rulesets. The resulting system provides a flexible toolbox that embeds Semantic Web data and ontologies in a fully declarative LP environment.

9 Conclusion

Answer Set Programming is a booming paradigm for declarative problem solving, which emerged from Logic Programming and Nonmonotonic Reasoning and has been deployed to a range of application areas. A number of answer set solvers are available which provide a variety of constructs and the features for problem modeling, helping the user to find formalizations of problems in a more natural and understandable manner. As for the Semantic Web, extensions of the basic ASP languages and formalisms have been developed, aiming at different goals. For example, to provide a formalism for combining rules and ontologies (e.g., dl-programs, $\mathcal{DL}+log$[114] and MKNF knowledge bases [95], conceptual logic programs [66], hybrid and guarded hybrid knowledge bases [28,65], and open answer set programming [67]; see [43,41] and references therein), or more general formalisms for accessing and interfacing data on the (Semantic) Web like HEX-programs, and systems like dlvhex, OntoDLV, and GiaBATA.

The interest in Answer Set Programming and significance of the underlying stable model semantics could be experienced at last year's edition of the International Logic Programming conference, which dedicated a (well-attended) special session to discuss the influence of stable model semantics on the field of logic programming. It witnessed ASP to be a vibrant area of research in which despite the advances and developments in the last years still a number of research challenges exist.

While the theory of ASP is well-developed and applications are expanding, the deployment of ASP to an industrial scale needs further efforts (cf. Nicola Leone's talk at LPNMR 2007).[22] Next generation answer set solvers must be developed which provide better support for the needs in practice.

Among these needs are complex data structures including lists, sets, records etc.; underneath, this calls for function symbols (recall that in Prolog, lists are special syntax function symbols) and a move beyond the Datalog fragment of logic programming. As mentioned in Section 7, function symbols have been largely banned because the quickly lead to undecidability. Only more recently, work on decidable classes of and prototype implementations of stable models semantics with function symbols has been carried out, including [119,16,12,15,20,116,51], and function symbols also increasingly attract attention as a modeling construct.

The class of ω-restricted programs [119] has been implemented on top of Smodels, and the recently presented class of finitely-ground programs in the DLV-Complex system on top of DLV [19,20], which aims at providing functions symbols in a decidable setting, giving support to lists and sets along with libraries for their manipulations.

[22] http://lpnmr2007.googlepages.com/nicola-lpnmr07.pdf

However, both system, models are always finite, which hinders modeling infinite processes and objects; classes like finitary programs [16], finitely recursive programs [12], FDNC-programs [116], and BD-programs [51] do not have this restriction, but lack implementations to date.

Related to this issue is incremental model building, which is of particular interest for applications in reasoning about actions and change: a model may describe the evolution of the world, which happens from one epoch to the next. Here, it is desired to build the model according to the evolution, step by step; this is, for instance, relevant for planning. Recent work [58] aims in this direction, giving a formal framework for incremental model building.

Another issue of relevance for practice is modularity in ASP, and to provide means for code reuse. While modularity has been recognized as an important aspect more than a decade ago [40], it has only found more recently increasing attention, cf. [26,72,100,121,9]; the use of macros [11] and templates [21] aims in this direction. Specifically for the Semantic Web context, a modular formalism with multiple non-monotonic logic programs [106] and the `MWeb` framework [2] have been conceived. The formalism in [106] allows to interlink web-accessible logic programs, i.e., logic programs may refer to logic programs that may refer to remote knowledge bases distributed on the Web. `MWeb` attempts to enhance the Semantic Web with the notions of scope and context for modular web rule bases, and pays attention to support knowledge hiding and the safe use of strong and weak negation, as well as to different reasoning modes.

However, most of these approaches reduce a system of modules (polynomially) into a single global program, or impose constraints regarding the use of recursion, resulting in limited expressiveness. The recent approach in [26], which improves [40], has no such constraints but the high expressiveness comes with high worst case complexity in general. Efficient algorithms and implementations of yet expressive, natural modular ASP frameworks are an interesting topic of research.

Concerning efficiency, of course also improvements to solvers for ordinary ASP are desirable. In the recent years, there has been a lot of work on optimizations based on program equivalence, triggered by the seminal paper [84], which introduced a notion of strong equivalence between non-monotonic logic programs that takes nonmonotonicity into account and led to a whole family of notions of equivalence, which may be utilized to rewrite a program into an equivalent one that can be see e.g. [61,38,37,123] for more on this issue. Another issue is non-ground ASP processing, in order to overcome the intrinsic grounding bottleneck in the two step architecture of most answer set solvers. Work on this is underway, and techniques for partial and lazy grounding (as used e.g. in [20,58,101,76]) are helpful; however, the grounding techniques of advanced AS solvers are highly sophisticated and in most case very effective. Interesting in this regard is also the work of [54], which defines answer sets for first-order theories that can be non-Herbrand models, in terms of a formula in second-order logic. While this avoids grounding, it remains to be seen whether this approach can be effectively implemented (by reducing, e.g., fragments of the formalisms to standard theorem provers).

Finally, for deployment on a larger scale, more software engineering tools and methodologies are needed. Compared to other programming languages, there is currently little support for programmers available, and rich ASP programming environments are lacking, which include debuggers, visualization, libraries etc. The *International Workshop on Software Engineering for Answer Set Programming (SEA)* [30] was initiated as a forum for researchers interested in these issues.[23] Given the work in progress, we may expect significant advances and improvements here in the near future.

In conclusion, though ASP has developed vigorously, there is still much ado in an exciting area of research for theory and applications.

Acknowledgments. We thank Minh Dao-Tran, Wolfgang Faber and Gerald Pfeifer for their valuable comments and corrections.

References

1. Alferes, J.J., Amador, R., May, W.: A general language for evolution and reactivity in the semantic web. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 101–115. Springer, Heidelberg (2005)
2. Analyti, A., Antoniou, G., Damásio, C.V.: A principled framework for modular web rule bases and its semantics. In: Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), AAAI Press, Menlo Park (2008)
3. Analyti, A., Antoniou, G., Damásio, C.V., Wagner, G.: Stable Model Theory for Extended RDF Ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 21–36. Springer, Heidelberg (2005)
4. Apt, K., Blair, H., Walker, A.: Towards a Theory of Declarative Knowledge. In: Minker [93], pp. 89–148
5. Apt, K., Bol, N.: Logic programming and negation: A survey. Journal of Logic Programming 19/20, 9–71 (1994)
6. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF databases. In: Franconi and Tessaris [56]
7. Asparagus homepage (2005), `http://asparagus.cs.uni-potsdam.de/`
8. Baader, F.: Description logics. In: Franconi and Tessaris [56]
9. Balduccini, M.: Modules and Signature Declarations for A-Prolog: Progress Report. In: de Vos and Schaub [30], `http://sea07.cs.bath.ac.uk/downloads/sea07-proceedings.pdf`
10. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
11. Baral, C., Dzifcak, J., Takahashi, H.: Macros, Macro Calls and Use of Ensembles in Modular Answer Set Programming. In: Etalle, S., Truszczyński, M. (eds.) ICLP 2006. LNCS, vol. 4079, pp. 376–390. Springer, Heidelberg (2006)
12. Baselice, S., Bonatti, P.A., Criscuolo, G.: On finitely recursive programs. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 89–103. Springer, Heidelberg (2007)
13. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Annals of Mathematics and Artificial Intelligence 12, 53–87 (1994)
14. Bidoit, N.: Negation in rule-based database languages: A survey. Theor. Comput. Sci. 78(1), 3–83 (1991)
15. Baselice, S., Bonatti, P.A.: Composing normal programs with function symbols. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 425–439. Springer, Heidelberg (2008)

[23] `http://sea07.cs.bath.ac.uk/`

16. Bonatti, P.A.: Reasoning with infinite stable models. Artificial Intelligence 156(1), 75–111 (2004)
17. Buccafurri, F., Faber, W., Leone, N.: Disjunctive logic programs with inheritance. Theory and Practice of Logic Programming 2(3) (2002)
18. Buccafurri, F., Leone, N., Rullo, P.: Strong and Weak Constraints in Disjunctive Datalog. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS (LNAI), vol. 1265, pp. 2–17. Springer, Heidelberg (1997)
19. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. Annals of Mathematics and Artificial Intelligence 50(3-4), 333–361 (2007)
20. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable Functions in ASP: Theory and Implementation. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 407–424. Springer, Heidelberg (2008)
21. Calimeri, F., Ianni, G.: Template programs for Disjunctive Logic Programming: An operational semantics. AI Communications 19(3), 193–206 (2006)
22. Chen, W., Kifer, M., Warren, D.S.: Hilog: A foundation for higher-order logic programming. Journal of Logic Programming 15(3), 187–230 (1993)
23. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)
24. Viegas Damásio, C., Analyti, A., Antoniou, G., Wagner, G.: Supporting open and closed world reasoning on the web. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, pp. 149–163. Springer, Heidelberg (2006)
25. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys 33(3), 374–425 (2001)
26. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular nonmonotonic logic programming revisited. In: Hill, P., Warren, D. (eds.) Proceedings 25th International Conference on Logic Programming (ICLP 2009). LNCS, vol. 5649, pp. 145–159. Springer, Heidelberg (2009)
27. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: On representational issues about combinations of classical theories with nonmonotonic rules. In: Lang, J., Lin, F., Wang, J. (eds.) KSEM 2006. LNCS, vol. 4092, pp. 1–22. Springer, Heidelberg (2006)
28. de Bruijn, J., Pearce, D., Polleres, A., Valverde, A.: Quantified equilibrium logic and hybrid rules. In: Marchiori, M., Pan, J.Z., de Marie, C.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 58–72. Springer, Heidelberg (2007)
29. de la Banda, M.G., Pontelli, E. (eds.): ICLP 2008. LNCS, vol. 5366. Springer, Heidelberg (2008)
30. de Vos, M., Schaub, T. (eds.): Informal Proceedings of the 1st International Workshop on Software Engineering for Answer Set Programming, Tempe, AZ, USA (May 2007), http://sea07.cs.bath.ac.uk/downloads/sea07-proceedings.pdf
31. Dix, J.: A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties. Fundam. Inform. 22(3), 227–255 (1995)
32. Dix, J.: A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties. Fundam. Inform. 22(3), 257–288 (1995)
33. Drabent, W., Eiter, T., Ianni, G., Krennwallner, T., Lukasiewicz, T., Małuszyński, J.: Hybrid reasoning with rules and ontologies. In: Bry, F., Małuszyński, J. (eds.) Semantic Techniques for the Web: The REWERSE perspective, ch. 1. LNCS, vol. 5500, p. 50. Springer, Heidelberg (to appear, 2009)
34. Eiter, T.: Answer set programming for the Semantic Web (tutorial). In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 23–26. Springer, Heidelberg (2007); Slides, http://www.dcc.fc.up.pt/iclp07/eiter.pdf

35. Eiter, T., Faber, W., Fink, M., Woltran, S.: Complexity results for answer set programming with bounded predicate arities and implications. Annals of Mathematics and Artificial Intelligence 51(2-4), 123–165 (2007)
36. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Declarative problem-solving using the DLV system. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 79–103. Kluwer Academic Publishers, Dordrecht (2000)
37. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS, vol. 2923, pp. 87–99. Springer, Heidelberg (2003)
38. Eiter, T., Fink, M., Woltran, S.: Semantical Characterizations and Complexity of Equivalences in Answer Set Programming. ACM Trans. Comput. Log. 8(3), Article 17 (53 + 11) (2007)
39. Eiter, T., Gottlob, G.: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. Annals of Mathematics and Artificial Intelligence 15(3/4), 289–323 (1995)
40. Eiter, T., Gottlob, G., Veith, H.: Modular Logic Programming and Generalized Quantifiers. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 290–309. Springer, Heidelberg (1997)
41. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and Ontologies for the Semantic Web. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web. LNCS, vol. 5224, pp. 1–53. Springer, Heidelberg (2008); Slides, http://rease.semanticweb.org/
42. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R.: Well-founded semantics for description logic programs in the Semantic Web. Technical Report INFSYS RR-1843-09-01, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria (March 2009)
43. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with Description Logics for the Semantic Web. Artificial Intelligence 172(12-13), 1495–1539 (2008)
44. Eiter, T., Ianni, G., Polleres, A., Schindlauer, R., Tompits, H.: Reasoning with rules and ontologies. In: Barahona, P., Bry, F., Franconi, E., Henze, N., Sattler, U. (eds.) Reasoning Web 2006. LNCS, vol. 4126, pp. 93–127. Springer, Heidelberg (2006)
45. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK, August 2005, pp. 90–96 (2005)
46. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic web reasoning. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 273–287. Springer, Heidelberg (2006)
47. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H., Wang, K.: Forgetting in managing rules and ontologies. In: IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006), Hongkong, pp. 411–419. IEEE Computer Society, Los Alamitos (2006); preliminary version at ALPSWS 2006
48. Eiter, T., Leone, N., Pearce, D.: Assumption Sets for Extended Logic Programs. In: Gerbrandy, J., Marx, M., de Rijke, M., Venema, Y. (eds.) JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday. Amsterdam University Press (1999), http://www.kr.tuwien.ac.at/staff/eiter/et-archive/jfak.pdf

49. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. In: Dubois, D., Welty, C., Williams, M.-A. (eds.) Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, British Columbia, Canada, pp. 141–151. Morgan Kaufmann, San Francisco (2004)
50. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Well-founded semantics for description logic programs in the Semantic Web. In: Antoniou, G., Boley, H. (eds.) RuleML 2004. LNCS, vol. 3323, pp. 81–97. Springer, Heidelberg (2004)
51. Eiter, T., Šimkus, M.: Bidirectional answer set programs with function symbols. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). AAAI Press, Menlo Park (2009)
52. Faber, W.: Enhancing Efficiency and Expressiveness in Answer Set Programming Systems. PhD thesis, Institut für Informationssysteme, Technische Universität Wien (2002)
53. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the DLV system. Theory and Practice of Logic Programming 8(5-6), 545–580 (2008)
54. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Veloso, M.M. (ed.) IJCAI, pp. 372–379 (2007)
55. Ferraris, P., Lifschitz, V.: Mathematical foundations of answer set programming. In: We Will Show Them! Essays in Honour of Dov Gabbay, vol. 1, pp. 615–664. College Publications (2005)
56. Franconi, E., Tessaris, S. (eds.): Reasoning Web 2009. LNCS. Springer, Heidelberg (2009)
57. Friedrich, G., et al.: Model-based repair of web service processes. Technical Report 2008/001, ISBI research group, University of Klagenfurt (2008), http://test-informations.info/
58. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an Incremental ASP Solver. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 190–205. Springer, Heidelberg (2008)
59. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczynski, M.: The First Answer Set Programming System Competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
60. Gelfond, M.: Representing Knowledge in A-Prolog. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS (LNAI), vol. 2408, pp. 413–451. Springer, Heidelberg (2002)
61. Gelfond, M.: Answer sets. In: van Harmelen, B.P.F., Lifschitz, V. (eds.) Handbook of Knowledge Representation, ch. 7, pp. 285–316. Elsevier, Amsterdam (2008)
62. Gelfond, M., Leone, N.: Logic programming and knowledge representation - the a-prolog perspective. Artificial Intelligence 138(1-2), 3–38 (2002)
63. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Proceedings Fifth Intl. Conference and Symposium Logic Programming, pp. 1070–1080. MIT Press, Cambridge (1988)
64. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
65. Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., Nieuwenborgh, D.V.: Guarded hybrid knowledge bases. Theory and Practice of Logic Programming 8(3), 411–429 (2008)
66. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Conceptual logic programs. Annals of Mathematics and Artificial Intelligence 47(1-2), 103–137 (2006)
67. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Open answer set programming for the Semantic Web. J. Applied Logic 5(1), 144–169 (2007)
68. Heymans, S., Toma, I.: Ranking Services Using Fuzzy HEX-Programs. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 181–196. Springer, Heidelberg (2008)

69. Hoehndorf, R., Loebe, F., Kelso, J., Herre, H.: Representing default knowledge in biomedical ontologies: Application to the integration of anatomy and phenotype ontologies. BMC Bioinformatics 8(1), 377 (2007)
70. Hustadt, U., Motik, B., Sattler, U.: Reducing SHIQ-description logic to disjunctive datalog programs. In: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, Canada, pp. 152–162 (2004)
71. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: A Rule System for Querying Persistent RDFS Data. In: Arroyo, L., Traverso, P. (eds.) The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Greece. LNCS, vol. 5554, pp. 857–862. Springer, Heidelberg (2009)
72. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity Aspects of Disjunctive Stable Models. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 175–187. Springer, Heidelberg (2007)
73. Kowalski, R.: Algorithm = Logic + Control. Commun. ACM 22(7), 424–436 (1979)
74. Lee, J.: A model-theoretic counterpart of loop formulas. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI, pp. 503–508. Professional Book Center (2005)
75. Lee, J., Lifschitz, V.: Loop Formulas for Disjunctive Logic Programs. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 451–465. Springer, Heidelberg (2003)
76. Lef'evre, C., Nicolas, P.: Integrating grounding in the search process for answer set computing. In: ASPOCP: Answer Set Programming and Other Constraint Paradigms, pp. 89–103 (2008)
77. Leone, N., Faber, W.: The DLV project: A tour from theory and research to applications and market. In: de la Banda and Pontelli [29], pp. 53–68
78. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
79. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Information and Computation 135(2), 69–112 (1997)
80. Lifschitz, V.: Nonmonotonic databases and epistemic queries. In: Proceedings IJCAI 1991, pp. 381–386 (1991)
81. Lifschitz, V.: Answer set planning. In: ICLP, pp. 23–37 (1999)
82. Lifschitz, V.: Answer Set Programming and Plan Generation. Artificial Intelligence 138, 39–54 (2002)
83. Lifschitz, V.: Twelve definitions of a stable model. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 37–51. Springer, Heidelberg (2008)
84. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. 2(4), 526–541 (2001)
85. Lifschitz, V., Razborov, A.A.: Why are there so many loop formulas? ACM Trans. Comput. Log. 7(2), 261–268 (2006)
86. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: Van Hentenryck, P. (ed.) Proceedings of the 11th International Conference on Logic Programming (ICLP 1994), Santa Margherita Ligure, Italy, pp. 23–37. MIT Press, Cambridge (1994)
87. Lin, F., Zhao, Y.: ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In: AAAI/IAAI, p. 112 (2002)
88. Lukasiewicz, T.: Probabilistic description logic programs. Int. J. Approx. Reasoning 45(2), 288–307 (2007)
89. Lukasiewicz, T., Straccia, U.: Description logic programs under probabilistic uncertainty and fuzzy vagueness. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS, vol. 4724, pp. 187–198. Springer, Heidelberg (2007)

90. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in dlv: Implementation, evaluation, and comparison to qbf solvers. J. Algorithms 63(1-3), 70–89 (2008)
91. Marek, V.W., Truszczyński, M.: Autoepistemic Logic. Journal of the ACM 38(3), 588–619 (1991)
92. Marek, V.W., Truszczyński, M.: Stable Models and an Alternative Logic Programming Paradigm. In: Apt, K., Marek, V.W., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm – A 25-Year Perspective, pp. 375–398. Springer, Heidelberg (1999)
93. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Washington (1988)
94. Minker, J.: Logic and Databases: A 20 Year Retrospective. In: Pedreschi, D., Zaniolo, C. (eds.) LID 1996. LNCS, vol. 1154, pp. 3–57. Springer, Heidelberg (1996)
95. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI 2007, pp. 477–482 (2007)
96. Niemelä, I.: Logic Programming with Stable Model Semantics as Constraint Programming Paradigm. Annals of Mathematics and Artificial Intelligence 25(3–4), 241–273 (1999)
97. Niemelä, I. (ed.): Language Extensions and Software Engineering for ASP. Technical Report WP3, Working Group on Answer Set Programming (WASP), IST-FET-2001-37004 (September 2005), http://www.tcs.hut.fi/Research/Logic/wasp/wp3/wasp-wp3-web/
98. Nieuwenborgh, D.V., Cock, M.D., Vermeir, D.: Computing Fuzzy Answer Sets Using dlvhex. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 449–450. Springer, Heidelberg (2007)
99. Nieuwenborgh, D.V., Eiter, T., Vermeir, D.: Conditional Planning with External Functions. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 214–227. Springer, Heidelberg (2007)
100. Oikarinen, E., Janhunen, T.: Achieving compositionality of the stable model semantics for Smodels programs. Theory and Practice of Logic Programming 8(5–6), 717–761 (2008)
101. Palù, A.D., Dovier, A., Pontelli, E., Rossi, G.: Gasp: Answer set programming with lazy grounding. In: LaSh 2008: Logic And Search - Computation of structures from declarative descriptions (2008)
102. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Longman, Amsterdam (1994)
103. Pearce, D.: Equilibrium logic. Annals of Mathematics and Artificial Intelligence 47(1-2), 3–41 (2006)
104. Pearce, D., Valverde, A.: Quantified equilibrium logic and foundations for answer set programs. In: de la Banda and Pontelli [29], pp. 546–560
105. Polleres, A.: From SPARQL to rules (and back). In: Proceedings of the 16th International Conference on World Wide Web (WWW), pp. 787–796. ACM, New York (2007)
106. Polleres, A., Feier, C., Harth, A.: Rules with Contextually Scoped Negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
107. Pontelli, E., Son, T.C., Baral, C.: A framework for composition and inter-operation of rules in the semantic web. In: Eiter, T., Franconi, E., Hodgson, R., Stephens, S. (eds.) RuleML, pp. 39–50. IEEE Computer Society, Los Alamitos (2006)
108. Provetti, A., Son, T.C. (eds.): Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, Proceedings of the 1st Intl. ASP 2001 Workshop, Stanford (March 26-28, 2001)
109. Przymusinski, T.C.: On the Declarative Semantics of Deductive Databases and Logic Programs. In: Minker [93], pp. 193–216

110. Rainer, A.: Web Service Composition under Answer Set Programming. In: Proc. KI 2005 Workshop "Planen, Scheduling und Konfigurieren, Entwerfen", PuK 2005 (2005)
111. Reiter, R.: On Closed-World Databases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 55–76. Plenum Press, New York (1978)
112. Reiter, R.: A Logic for Default Reasoning. Artificial Intelligence 13(1–2), 81–132 (1980)
113. Ricca, F., Gallucci, L., Schindlauer, R., Dell'armi, T., Grasso, G., Leone, N.: OntoDLV: An ASP-based System for Enterprise Ontologies. Journal of Logic and Computation (2008), doi:10.1093/logcom/exn042
114. Rosati, R.: $\mathcal{DL}$+*log*: Tight Integration of Description Logics and Disjunctive Datalog. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning KR 2006, pp. 68–78. AAAI Press, Menlo Park (2006)
115. Ross, K.A.: Modular stratification and magic sets for datalog programs with negation. Journal of the ACM 41(6), 1216–1266 (1994)
116. Šimkus, M., Eiter, T.: FDNC: Decidable non-monotonic disjunctive logic programs with function symbols. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 514–530. Springer, Heidelberg (2007); Extended Paper to appear in ACM Trans. Computational Logic
117. Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comput. Sci. 3(1), 1–22 (1976)
118. Swift, T.: Deduction in Ontologies via ASP. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 275–288. Springer, Heidelberg (2003)
119. Syrjänen, T.: Omega-restricted logic programs. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS, vol. 2173, pp. 267–279. Springer, Heidelberg (2001)
120. Syrjänen, T., Niemelä, I.: The smodels system. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS, vol. 2173, pp. 434–438. Springer, Heidelberg (2001)
121. Tari, L., Baral, C., Anwar, S.: A Language for Modular Answer Set Programming: Application to ACC Tournament Scheduling. In: Proceedings of the 3rd International ASP 2005 Workshop, Bath, UK, July 2005. CEUR Workshop Proceedings, vol. 142, pp. 277–293. CEUR WS (2005)
122. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM 38(3), 620–650 (1991)
123. Woltran, S.: A common view on strong, uniform, and other notions of equivalence in answer-set programming. Theory and Practice of Logic Programming 8(2), 217–234 (2008)
124. Woltran, S. (ed.): Answer Set Programming: Model Applications and Proofs-of-Concept. Technical Report WP5, Working Group on Answer Set Programming (WASP), IST-FET-2001-37004 (July 2005), http://www.kr.tuwien.ac.at/projects/WASP/report.html

A Appendix: A DLV Specification for the Sudoku Problem

```
#maxint=9.

tab(X,Y,1) v tab(X,Y,2) v tab(X,Y,3) v
tab(X,Y,4) v tab(X,Y,5) v tab(X,Y,6) v
tab(X,Y,7) v tab(X,Y,8) v tab(X,Y,9)
   :- #int(X), 0 <= X, X <= 8, #int(Y), 0 <= Y, Y <= 8.

% Check rows and columns
:- tab(X,Y1,Z), tab(X,Y2,Z), Y1<>Y2.
:- tab(X1,Y,Z), tab(X2,Y,Z), X1<>X2.
```

```
% Check subtable
:- tab(X1,Y1,Z), tab(X2,Y2,Z), Y1 <> Y2,
   div(X1,3,W1), div(X2,3,W1),
   div(Y1,3,W2), div(Y2,3,W2).

:- tab(X1,Y1,Z), tab(X2,Y2,Z), X1 <> X2,
   div(X1,3,W1), div(X2,3,W1),
   div(Y1,3,W2), div(Y2,3,W2).

%Auxiliary: X divided by Y is Z
div(X,Y,Z) :- XminusDelta = Y*Z,
X = XminusDelta + Delta, Delta < Y.

% Table positions  X=0..8, Y=0..8
tab(0,1,6). tab(0,3,1). tab(0,5,4). tab(0,7,5).
tab(1,2,8). tab(1,3,3). tab(1,5,5). tab(1,6,6).
tab(2,0,2). tab(2,8,1). tab(3,0,8). tab(3,3,4).
tab(3,5,7). tab(3,8,6).
tab(4,2,6). tab(4,6,3).
tab(5,0,7). tab(5,3,9). tab(5,5,1). tab(5,8,4).
tab(6,0,5). tab(6,8,2).
tab(7,2,7). tab(7,3,2). tab(7,5,6). tab(7,6,9).
tab(8,1,4). tab(8,3,5). tab(8,5,8). tab(8,7,7).
```

B Appendix: Fixpoint Theorems of Knaster-Tarski and Kleene

Definition 18. *A complete lattice is a partially ordered set $(V, \leq)$ such that each subset $W \subseteq V$ has a least upper bound* $\sup(W)$ *and a greatest lower bound* $\inf(W)$.

Example 48. The partially ordered set $(V, \leq)$, where V is the set of all Herbrand interpretations of a program P and $\leq$ is set inclusion ($\subseteq$), is a complete lattice.

Definition 19. *An operator on a complete lattice $(V, \leq)$ is a mapping $T\colon V \to V$.*

Example 49. The T_P operator for a program P is an operator on Herbrand interpretations.

Definition 20. *An operator $T\colon V \to V$ on $(V, \leq)$ is* monotone*, if*

$$x \leq y \textit{ implies } T(x) \leq T(y)\ \forall x, y \in V \ .$$

Monotone operators have nice fixpoint properties.

Theorem 13 (Knaster-Tarski). *Any monotone operator T on a complete lattice $(V, \leq)$ has a least fixpoint $lfp(T)$, and*

$$lfp(T) = \inf(\{x \in V \mid T(x) \leq x\}) \ .$$

Example 50. The T_P operator for a (positive) program P is monotone.

A stronger theorem holds for continuous operators.

Definition 21. *A set $W \subseteq V$ is* directed, *if for each $x, y \in W$ there exists some $z \in W$ such that $x \leq z$ and $y \leq z$, where $(V, \leq)$ is a partial order.*

Definition 22. *An operator $T\colon V \to V$ on a complete lattice $(V, \leq)$ is* continuous, *if*

$$T(\sup(W)) = \sup(\{T(x) \mid x \in W\})$$

for every directed set $W \subseteq V$.

Intuitively, directedness models convergence (one can build a chain $x_0 < x_1 < \cdots$). It is not difficult to see that continuous operators are also monotone.

Example 51. The T_P operator is also continuous.

Theorem 14 (Kleene). *Any continuous operator T on a complete lattice $(V, \leq)$ has a least fixpoint, and*

$$\mathit{lfp}(T) = \sup(\{T^i \mid i \geq 0\}) \ ,$$

where $T^0 = \inf(V)$ and $T^{i+1} = T(T^i)$, for all integers $i \geq 0$.

Let $T^\infty = \sup(\{T^i \mid i \geq 0\})$. Note that if $T^i = T^{i-1}$ for some i, then $T^\infty = T^i$ holds; in particular, this is the case for T_P if the program P has no function symbols (given P is finite).

Remark. A weaker form of Kleene's Theorem holds for all monotone operators ($\mathit{lfp}(T)$ is constructible by a transfinite sequence T^α, for ordinals $\alpha \geq 0$).

Logical Foundations of XML and XQuery

Maarten Marx

ISLA, Universiteit van Amsterdam
The Netherlands

Abstract. XML is the underlying representation formalism of much web-data. Thus to reason about web-data essentially boils down to reasoning about data in XML format. In this course the students learn about the main languages for querying XML data: XPath and XQuery. The course contains both theoretical work and practical examples.

1 Introduction

These lecture notes are based on four articles about querying XML with XPath and XQuery. The first paper shows how XPath is rooted in modal and temporal logic. It also contains an introduction to real query writing for logicians. We formulate the binary until connective from temporal logic in a number of ways and show the effect of the particular formulation on performance.

The second paper is purely theoretical It gives mathematical-logical results about the family of XPath languages: on expressivity, axiomatizability and complexity. It also investigates suitable algebraic counterparts of different XPath languages. There is quite some overlap between these two papers. We left that to preserve the original structure of the two separate papers. Readers can conveniently skip repeating parts.

The last two papers are practically oriented. The third is on the process of turning textual documents with implicit structure into XML. We call this process exemelification. It shows how Extract-Transform-Load [52] techniques can be formulated declaratively in XSLT. Exemelification is often a preprocessing step in creating true Semantic Web content, and often taken for granted in that community. We discuss several information retrieval aspects of XML documents.

The last paper shows how beneficial XML can be for the social sciences and the humanities. It is an example of the emerging field of *Computational Social Science* [42]. Step by step we show the operationalization written in XQuery of a research question from political science.

2 Modal Logical Roots of XPath

This section is based on joint work with Loredana Afanasiev and Balder ten Cate [4].

2.1 Introduction

It has been proclaimed that *logic is the calculus of computer science* [32], but most applied computer scientists do not use logical formulas in a shape recognizable by logicians — at least not beyond propositional logic. In particular, this holds for *modal* *logic*. Even though theoretical computer scientists have created many different modal logics, few of them are used in practice. Until recently, it was fair to say that the *temporal logics* LTL and CTL were the best-known and most applied modal logics [14]. With the advent of XML and its associated querying and processing languages, it seems this position has been taken over by the XML document navigation language XPath.

XPath is a core fragment of the XML query language XQuery and transformation language XSLT. All modern web browsers (e.g., *Internet Explorer* and *Firefox*) are able to process XPath expressions, and most serious web programmers write XPath expressions on a regular basis (whether they are conscious of it or not). Even though the current XPath manual contains 530 pages [38], the logical core of the language is small and closely resembles modal logic. Thus, XPath is arguably the best known and most applied modal logic today.

In this paper we give a gentle introduction to the XPath language from a logician's point of view. We discuss how its connections to modal logic have led to a series of results characterizing the expressive power of this language and the complexity of various computational tasks, such as query evaluation and query containment testing.

2.2 Modal Logic, Temporal Logic, and PDL

Nowadays, modal logic is mostly viewed as a language for describing graph-like structures. This 'semantic' view of modal logic stands in contrast to the more traditional 'syntactic' view, according to which modal logic is just a propositional language enriched with two non truth-functional operators $\Diamond$ and $\Box$.

As Hans Kamp recently described it[1], one of the most important steps in the historical development of modal logic was to break out of the rigid box/diamond format. Kamp himself made the first step in 1968 by introducing *until* [36]. This "*binary* modalities" makes it possible, in the context of temporal logic, to express things like "A will be true at some point in the future and, until then, B is true" ($until(A, B)$ for short). *Until* make temporal logic very expressive. In fact, on time flows isomorphic to the natural numbers, temporal logic with *until* and its mirror image *since* is as expressive as first-order logic, a result that has become known as *Kamp's Theorem* [36]. However, what Kamp described as a real watershed in modal logic was the invention of Propositional Dynamic Logic (PDL) in the 1970s [24]. PDL is a modal logic with infinitely many modalities (or, 'programs'), that form an algebra generated by atomic programs and formula tests using the operations of *composition*, *union* and *reflexive transitive closure*, familiar from regular expressions.

Since PDL turns out to be closely related to the XML language XPath, let us explain its syntax and semantics in a bit more detail. PDL expressions are interpreted over *node labelled directed multi-graphs*, i.e., directed graphs with several relations, whose nodes are labelled.[2] There are two types of expressions in PDL: *state formulas* ($\phi, \psi, \dots$) describing properties of points (or, 'states') in the graph, and *program formulas* ($\alpha, \beta, \dots$)

[1] A 15 minute presentation about the history of modal logic, on the occasion of the launch of the *Handbook of Modal Logic*, on December 18, 2006, in the Oost Indisch Huis, Amsterdam.

[2] Usually, it is assumed that nodes can satisfy more than one label, but this is not important for us.

describe ways of traversing it (more formally, they define binary relations over the domain of the graph). They are defined by mutual recursion:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle\alpha\rangle\phi$$

$$\alpha ::= a \mid \phi? \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha^*$$

where p is a node label and a is any of the relations of the multi-graph. State formulas of the form $\langle\alpha\rangle\phi$ are interpreted as "some point reachable from the current point by α satisfies ϕ", and state formulas of the form $[\alpha]\phi$, dually, are interpreted as "every point reachable from the current point by α satisfies ϕ". The atomic program formula a is interpreted as "move to a node that is a successor of the current point with respect to the relation a", and $\phi?$ is interpreted as "stay at the current node but check that ϕ is true there". The program operations $/$, $\cup$ and $*$ are interpreted as composition, union and reflexive-transitive closure.

To illustrate the expressive power of PDL, observe that, on the natural numbers with the successor relation next, $until(A, B)$ can be expressed in PDL as

$$\langle(\mathsf{next}/B?)^*/\mathsf{next}\rangle A,$$

that is, "it is possible to reach a node satisfying A by the following program: do a finite number of successor steps followed by tests on B, and end with one more successor step".

PDL is quite well understood as a logical language. For instance, we know the computational complexity of various tasks involving PDL formulas such as *model checking* and *entailment*.

Fact 1 (Complexity of PDL model checking). Given a finite node labelled directed multi-graph G with a designated state s, and given a PDL formula ϕ, it can be tested in time $\mathcal{O}(|G|\cdot|\psi|)$ whether ϕ is true at s in G. Moreover, this problem is PTIME-complete.

Fact 2 (Complexity of PDL entailment). Given PDL formulas ϕ, ψ, it can be tested in time $2^{\mathcal{O}(|\phi|+|\psi|)}$ whether ϕ implies ψ (i.e., whether in every model, every state satisfying ϕ also satisfies ψ.) Moreover, this problem is EXPTIME-complete.

For more information, see e.g. [9,15].

2.3 XML and Sibling Ordered Trees

XML is a standard format for representation and exchange of *semi-structured data* on the internet. Semi-structured data is, roughly, data that is not structured enough to fit nicely in tables. While in the relational database model, the fundamental datastructure is that of a table, the fundamental datastructure in XML is that of (finite) *node labelled sibling ordered trees*: node labelled finite trees in which the children of each node are linearly ordered. This is a very natural datastructure, and it appears in many settings. For instance, the parse tree of a natural language sentence, or of an algebraic term, is a finite node labelled sibling ordered tree. An example of an XML document is given in Figure 1.

The two most important languages for manipulating XML data are the *query language* XQuery and the *transformation language* XSLT. Here is an example of an XQuery query:

```
 <note date='10-Nov-2006'>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget about <i>me</i>
         this weekend!</body>
</note>
```

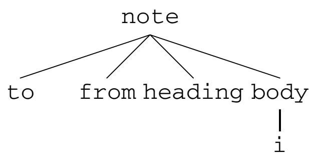


Fig. 1. Example XML document with skeleton

```
for $x in doc("notes.xml")//note
where $x/@date='10-Nov-2006'
return <recipient>$x/to</recipient>
```

This query could be run on a document containing a collection of notes, where each note is of the form described in Figure 1. The query then returns all recipients of notes sent on 10 November 2006, structured as XML using the `<recipient>` tag. The italic sub expressions are called *path expressions*. They select elements from the document. *XPath* is the language of these path expressions. It is a common fragment of XQuery and XSLT, and it is the topic of this paper.

In what follows, we will make one important simplifying assumption: when considering XML documents, we will abstract away from the actual data in the XML document (i.e., the text in-between the tags, and the attribute-value information). What is left is the 'skeleton' of the XML document: the hierarchical structure and the XML tag labels of the nodes. This allows us to view XML documents as nothing more than *node labelled sibling ordered trees*, where the node labels are the XML tags. In the case of Figure 1, the skeleton is precisely the picture on the right (including the information of the sibling order).

2.4 XPath 1.0 and Its Navigational Core

XPath 1.0 was introduced as a language for addressing parts of an XML document. Inspired by Unix paths, its expressions describe ways to travel through an XML document. XPath 1.0 is a very rich language, in fact its technical specifications are about 30 pages long [13]. Since, in this paper, we abstract away from the data content of XML documents and consider only the 'skeleton', we can disregard much of the functionality of XPath 1.0. What is left is the *navigational core*, *Core XPath 1.0*, which was formally defined in [27]. Surprisingly, it turns out that Core XPath 1.0 is almost identical to PDL. The only real differences are the following:

– While PDL in general is interpreted over arbitrary directed multi-graphs, in the case of Core XPath 1.0 they are *finite sibling ordered trees*, i.e., XML documents. Correspondingly, there are four atomic programs, corresponding to the four basic moves in the tree: child, parent, previous-sibling and next-sibling. The node labels are simply the XML tags.
– The use of the *reflexive transitive closure* operator $*$ is restricted to atomic programs.

Besides these, there are some minor differences concerning notation and terminology. For instance, the 'program formulas' of PDL are called 'path expressions' in XPath, and 'state formulas' are called 'node tests' or 'filter expressions'.

In presenting the syntax of Core XPath 1.0 below, we will cheat a bit: we use a slightly different notation, to emphasize the connection with PDL. It should be noted, though, that there are linear translations between the two. The syntax of Core XPath 1.0 is as follows:

Path expressions (defining binary relations over the domain of the tree):

$$\begin{aligned}\alpha ::= {} & \mathsf{child} \mid \mathsf{parent} \mid \mathsf{previous\text{-}sibling} \mid \mathsf{next\text{-}sibling} \mid \\ & \mathsf{child}^* \mid \mathsf{parent}^* \mid \mathsf{previous\text{-}sibling}^* \mid \mathsf{next\text{-}sibling}^* \mid \\ & \mathsf{self} \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi]\end{aligned}$$

Node expressions (defining sets of nodes):

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle\alpha\rangle \qquad \text{(for } p \text{ an XML tag)}$$

Two constructs require explanation, namely $\alpha[\phi]$ and $\langle\alpha\rangle$. The first is interpreted as a 'filter': $\alpha[\phi]$ defines the subrelation of α containing all pairs (x, y) of which y satisfies ϕ. The node expression $\langle\alpha\rangle$ simply expresses that the node under evaluation is in the domain of the relation α. Thus, for example, the path expression $\mathsf{child}[\langle\mathsf{child}\rangle]$ — in the official XPath notation `child::*[child::*]` — says "go to a child that is not a leaf". Likewise, the path expression $\mathsf{child}[\mathit{note} \wedge \neg\langle\mathsf{child}[\mathit{body}]\rangle]$ — in the official XPath notation `child::note[not(child::body)]`, abbreviated as `note[not(body)]` — says "go to a child with tag *note* that has no child with tag *body*". Incidentally, note that, while PDL supports expressions of the form $\langle\alpha\rangle\phi$, these can be expressed in Core XPath 1.0 as $\langle\alpha[\phi]\rangle$.

In practice, an XPath path expression α is used as follows: it is evaluated at a specified context node x of a document D (typically the root), and yield a *set of nodes* $\mathrm{RESULT}_{D,x}(\alpha)$, consisting of those nodes of D that can be reached from x by executing α. The XQuery query given in Section 2.3 is a good example of this use of path expressions.

Complexity. The close relation between Core XPath 1.0 and PDL has been exploited to obtain complexity results for various tasks involving XPath expressions. For instance, the following results can be obtained in this way:

Fact 3 (Query evaluation complexity). Given a Core XPath 1.0 path expression α, an XML document D and a context node x, $\mathrm{RESULT}_{D,x}(\alpha)$ can be computed in linear time.

Fact 4 (Query containment complexity). The following problem is EXPTIME-complete: given Core XPath 1.0 path expressions α and β, does $\mathrm{RESULT}_{D,x}(\alpha) \subseteq \mathrm{RESULT}_{D,x}(\beta)$ hold for all D and x?

Fact 3 follows directly from Fact 1. For Fact 4 this is not the case. It does not follow directly from Fact 2 because the class of structures is different (an implication between two formulas may hold on trees but not on arbitrary structures). Nevertheless, essentially the same (automata-theoretic) techniques apply.

Expressive power. One way to measure the expressive power of a query language is by relating it to the expressive power of the logical language we know best: first-order logic. Edgar Codd proved for instance, within the framework of the relational database model, that *relational algebra* has the same expressive power as first-order logic. Since our XML documents are node labelled sibling ordered trees, we consider a first-order language that has binary predicates $<_V$ and $<_H$ for the *descendant* and *following sibling* relations, and a unary predicate for each XML tag. Call this language FO_{tree}.

The path expressions of Core XPath 1.0 can be naturally compared to FO_{tree}-formulas with two free variables: a variable x standing for the context node and another variable y standing for the retrieved node. We say that a path expression α is equivalent to an FO_{tree}-formula $\phi(x, y)$ if for all XML documents D and nodes d, e, it is the case that $e \in \text{RESULT}_{D,d}(\alpha)$ if and only if $D \models \phi(d, e)$. Likewise, the node expressions of Core XPath 1.0 can be naturally compared to FO_{tree}-formulas with one free variable.

Fact 5. Core XPath 1.0 is a *proper* fragment of FO_{tree}.

The easy half of this results says that Core XPath 1.0 is a fragment of FO_{tree}. This can be shown by means of a linear translation from Core XPath 1.0 path expressions to FO_{tree}-formulas in two free variables and from Core XPath 1.0 node expressions to FO_{tree}-formulas in one free variable. The more difficult half of the result says that there are FO_{tree}-formulas that are not equivalent to any Core XPath 1.0 node or path expression. The *until* operator from Section 2.2 can be used here. In the case of XML documents, there are four natural versions of *until*: upward, downward, leftward and rightward. For instance, the downward version of until talks about descendants of a node: $until_\downarrow(A, B)$ holds at a node x iff $\exists y.(x <_V y \land A(y) \land \forall z(x <_V z \land z <_V y \rightarrow B(z)))$. Notice how this immediately shows that $until_\downarrow$ is FO_{tree}-definable. Likewise for the other versions of *until*. On the other hand, none of them can be expressed in Core XPath 1.0. A proof is given in [48], along the following lines: there is a (linear) translation from Core XPath 1.0 node expressions to first-order formulas containing only two[3] variables. On the other hand, $until_\downarrow(A, B)$ and the other versions cannot be expressed with less than three variables (the latter can be shown using a variant of Ehrenfeucht-Fraïssé games [17]).

Incidentally, recall from Section 2.2 that *until* queries can be expressed in PDL. However, this requires the use of $*$ on non-atomic expressions (in particular, on expressions of the form $\alpha[\phi]$ with $\alpha \in \{\mathsf{child}, \mathsf{parent}, \mathsf{previous\text{-}sibling}, \mathsf{next\text{-}sibling}\}$.

A precise characterization of the expressive power of Core XPath 1.0 in terms of a two-variable fragment of FO_{tree} was given in [48], based on similar results for temporal logics without *until* [19].

2.5 Two Extensions of Core XPath 1.0

Soon after XPath 1.0 was introduced, users found that they needed more expressive power. We will discuss two directions in which one can expand XPath. On the one hand, the connection between XPath and PDL naturally suggests extending XPath with

[3] The translation goes to the expansion of FO_{tree} with predicates for the child and the next-sibling relation. These are definable in FO_{tree}, but the definition needs three variables.

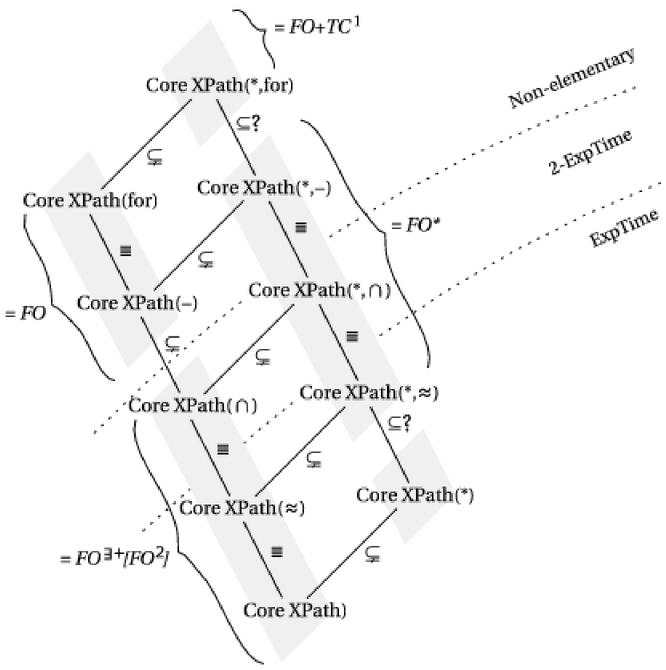


Fig. 2. Hierarchy of extensions of Core XPath 1.0

an unrestricted *transitive closure* operator. On the other hand, the W3C has introduced the 2.0 version of XPath, which extends XPath 1.0 in a different direction, namely with intersection, complementation, and quantified variables.

A summary of most of the results described in this section (and more) can be found in Figure 2. For a more detailed explanation of this diagram, cf. [62].

Regular XPath: Adding the full transitive closure operator. The connection between XPath and PDL described above naturally suggests extending XPath 1.0 with a reflexive transitive closure operator that operates on arbitrary (not only atomic) path expressions. Indeed, there are various reasons for extending the language in this way [1]. The extension of Core XPath 1.0 in which $*$ can be applied to arbitrary path expressions is called *Regular XPath*.

Regular XPath is much more expressive than Core XPath. For instance, the *until* queries discussed in Section 2.4 can be expressed in it. As a matter of fact,

Fact 6. Regular XPath strictly extends FO_{tree} in expressive power.

There are two sides to this result. Firstly, it says that Regular XPath can express all FO_{tree}-definable properties of nodes and binary relation. This was proved in [47] by an adaptation of a proof of Kamp's Theorem in temporal logic (cf. Section 2.2). The second side of the Fact 6 is that there are node expressions and path expressions of Regular XPath that have no FO_{tree} equivalent. A simple example is the path expression

$(\mathsf{child}/\mathsf{child})^*$, which says "make an even number of child steps". It is well known that such properties cannot be expressed in first-order logic, and it can be proved formally using Ehrenfeucht-Fraïssé games [17].

Even though Regular XPath is more expressive than FO_{tree} there are FO_{tree}-formulas for which the smallest equivalent Regular XPath expression is *much* longer. In fact, non-elementarily longer. (a function $f : \mathbb{N} \to \mathbb{N}$ is said to be non-elementary if it grows faster than any tower of exponentials of fixed height, as in $2^{(2^{(\cdots^{n})})}$).

No precise characterization is known of the expressive power of Regular XPath. There is, however, a characterization of the extension of Regular XPath with $loop$. The $loop$ construct allows us to test whether a node is related to itself by a path expression. Formally, a node d satisfies $loop(\alpha)$, for α any path expression, iff (d, d) belongs to the relation α. The extension Regular XPath with $loop$ is called *Regular XPath*$^{\approx}$. The following example illustrates the convenience of having $loop$ in the language.

Example 1. Consider the following properties of nodes in a tree: "*having an even number of descendants*". This property can be expressed by a Regular XPath$^{\approx}$ node expression. To see this, first, let next be shorthand for

$$\mathsf{child}[\neg\langle\mathsf{previous\text{-}sibling}\rangle] \ \cup \ \mathsf{self}[\neg\langle\mathsf{child}\rangle]/(\mathsf{parent}[\neg\langle\mathsf{next\text{-}sibling}\rangle])^*/\mathsf{next\text{-}sibling}$$

which defines the successor relation in the depth-first left-to-right ordering of the tree. A node satisfies $loop\big((\mathsf{next}/\mathsf{next})^*[\neg\langle\mathsf{child}\rangle]/(\mathsf{parent}[\neg\langle\mathsf{next\text{-}sibling}\rangle])^*\big)$ iff it has an even number of descendants.

As a matter of fact, it is possible to express the same property in Regular XPath without the use of $loop$. However, the solution is much less straightforward. The reader may consider it an exercise to find the right expression.

The expressive power of Regular XPath$^{\approx}$ can be characterized in terms of a natural extension of FO_{tree}. Let FO^*_{tree} be the extension of FO_{tree} in which, for each formula $\phi(x, y)$ with exactly two free variables, $\phi^*(x, y)$ is also allowed as a formula, and it defines the transitive closure of the relation defined by $\phi(x, y)$.

Fact 7. Regular XPath$^{\approx}$ has the same expressive power as FO^*_{tree}.

Again, there is a non-elementary succinctness gap: there are queries that can be expressed exponentially more succinctly in FO^*_{tree} than in Regular XPath$^{\approx}$ [61].

Finally, let us say something about complexity. Given that Regular XPath$^{\approx}$ is much more expressive than Core XPath 1.0, it is natural to ask whether the complexity goes up as well. Surprisingly, this is not the case (up to a polynomial):

Fact 8. Query evaluation (in the sense of Fact 3) for Regular XPath$^{\approx}$ can still be performed in PTIME.

Fact 9. Query containment (in the sense of Fact 4) for Regular XPath$^{\approx}$ is still EXPTIME-complete.

Fact 8 follows again from known results about PDL [41] while Fact 9 is proved in [62].

Core XPath 2.0: Adding intersection, complementation and quantified variables When the W3C committee designed XPath 2.0, rather than adding the unrestricted reflexive transitive closure operator, they decided to extend XPath 1.0 in another direction. We concentrate again on the navigational core, Core XPath 2.0. It extends Core XPath 1.0 with the following new operations:

- An intersection operator on path expressions.
  This allows us to write path expressions of the form $\alpha \cap \beta$, with the semantics "select all nodes that can be reached from the current node both by α and by β".
- A complementation operator on path expressions.
  This allows us to write path expressions of the form $\alpha - \beta$, with the semantics "select all nodes that can be reached from the current node by α and not by β".
- Variables and quantifiers.
  This allows us to write path expressions of the form for \$i in $Path_1$ return $Path_2$, interpreted as "assign to the variable \$i some node reachable from the current node by $Path_1$, and perform $Path_2$ under this variable assignment".

With these additional operators, one can express for instance *until* queries. For instance, $until_{\downarrow}(A, B)$ can be expressed as follows (with child$^+$ shorthand for child/child*):

$$(\mathsf{child}^+[A]) \ - \ (\mathsf{child}^+[\neg B]/\mathsf{child}^+). \tag{1}$$

or, using quantified variables, for instance as follows:

$$\mathsf{for\ \$i\ in\ self\ return\ child}^+[\mathsf{A} \wedge \neg(\mathsf{parent}^+[\mathsf{B}]/\mathsf{parent}^+[\mathsf{self\ is\ \$i}])] \tag{2}$$

The XPath operator is tests for node-equality. Thus the test self is \$i is true if and only if the current node (indicated by the path self) equals the node stored in the variable \$i.

As a matter of fact, Core XPath 2.0 has full first-order expressive power [47]:

Fact 10. Core XPath 2.0 has the same expressive power as FO_{tree}.

This time, there is no difference in succinctness between the two languages. Indeed, there are linear translations between the two languages. This is because, basically, the quantified variables make Core XPath 2.0 a notational variant of FO_{tree}. This might seem like a good result (after all, being able to express something succinctly is a good thing). However, it also has negative consequences. In particular, Core XPath 2.0 inherits bad complexity results from FO_{tree}:

Fact 11 (Query evaluation complexity). Query evaluation (in the sense of Fact 3) for Core XPath 2.0 is PSPACE-complete.

Fact 12 (Query containment complexity). Query containment (in the sense of Fact 4) for Core XPath 2.0 is non-elementary hard. Even Core XPath 1.0 extended with the complementation operator or with a single quantified variable already has a non-elementary hard query containment problem, and for the extension of Core XPath 1.0 with the intersection operator alone it is 2EXPTIME-complete.

See [62] for more information. Note that Fact 11 is concerned with the *combined complexity* of the query evaluation problem, where the document and the path expression are both part of the input. Following Vardi's taxonomy [68], one could also consider the *data complexity* for Core XPath 2.0 query containment, where only the document is counted as part of the input and the path expression is assumed to be fixed. The data complexity of Core XPath 2.0 query evaluation is PTIME.

Putting things together, we see that Core XPath 2.0 is less expressive than Regular XPath$^{\approx}$, but has a higher computational complexity, due to the fact that it is more succinct.

Needless to say, the full XPath 2.0's syntax is much richer than this navigational fragment. In fact, its underlying data-model differs from that of XPath 1.0 in some respects, but that is beyond the scope of this paper. An excellent introduction is [38].

2.6 An Experiment in Formulation

Throughout this section, *until* queries have been our running example. We saw that they cannot be expressed in XPath 1.0, but they can be expressed in Regular XPath using the reflexive transitive closure operator, or in XPath 2.0 using either the complementation operator or quantified variables. In this section, we look how these ways of expressing queries compare when executed on real XQuery systems (recall that XPath expressions are mostly used inside XQuery queries or XSLT transformations).

We created a simple experiment in which we express the vertical until query *until(A,B)* (i.e, over the `child` relation) on XML-trees in a number of different styles. We ran these equivalent queries on a large XML document (46MB) and recorded the times. The outcomes show a large variation in processing times, varying from less than 10 to over a 1000 seconds and engine crashes.

It is difficult to draw conclusions from the experiment which can be translated into practical advice to engine developers, besides the fact that query optimization can be useful. But the results hint at a practical advice to the developers of XPath/XQuery: add the Kleene star to XPath. The Kleene style of writing the until query outperforms all others; on talks we gave the majority of the audience predicted it to be the winner of the three styles; it is a natural, procedural way of specifying the until query. *But it is very difficult to express in XQuery*, in particular when compared to the simple formulation (3) in Regular XPath.

The problem. Given a context node x, we want to compute the set of nodes that can be reached from x along vertical *until(A,B)* paths. In Section 2.4 and 2.5 we have already seen different styles of expressing this query.

The first style uses the Kleene star of Regular XPath (i.e., the reflexive transitive closure operator), thus we call it the *Kleene* style: return all nodes reachable from the context node along the path

$$(\texttt{child}[B])^*/\texttt{child}[A]. \tag{3}$$

The second style uses the path complementation operation of XPath 2.0. We call it the *Tarski* style, because it uses the operations of Tarski's relation algebra: return all nodes reachable from the context node along the relation (here we use `descendant` as an abbreviation of `child`/`child`*)

$$(\texttt{descendant}[A]) \;-\; (\texttt{descendant}[\neg B]/\texttt{descendant}). \tag{4}$$

The third way of expressing the query, and probably the simplest one for a logician, is by using first-order logic and the universal quantifier: for x the context node, return all y satisfying

$$x <_V y \wedge A(y) \wedge \forall z(x <_V z \wedge z <_V y \rightarrow B(z)). \tag{5}$$

Recall that $<_V$ is the descendant relation. In honour of Frege's role in the historical development of first-order logic, we call this the *Frege* style of writing the query. There are various ways of writing this first-order formula in XPath 2.0. We will come back to this in a moment.

All three formulations describe the same binary relation.

From problem to query. Our aim is to compare the three styles of expressing until queries, by running them on a number of XQuery engines for a concrete XML document, and recording the processing times to find differences in performance.

We use an XML document from the Michigan XQuery benchmark [55] of 46MB and 728000 nodes. All elements in the document have the same label and attributes. We use one numeric attribute that has randomly generated values from 0 to 3 to express the node conditions A and B. A is the predicate condition `[@aFour=3]` and B is the predicate condition `[@aFour=0]`.

In the queries below, we use the notation $//*$. This just means "go to any node below the current node".

The outcome of the query should describe the range of the binary relation. We set up the query so that there are 16 "start-nodes", each with many descendants. These nodes are all the nodes at depth five in the XML tree and all queries start with the following piece of XQuery code

```
let $start := doc("mbench.xml")//*[@aLevel=5] return,
```

or `for $start in doc("mbench.xml")//*[@aLevel=5] return`

The query outcomes are sequences of elements that stand in the specified relation to the start-nodes. The results are ordered in the document order.

In this way, we came up with 7 equivalent queries U1–U7, specified in the Appendix on page 155. U1 and U2 are in the Kleene style and in the Tarski style, respectively. Since there are many natural ways of expressing the Frege style in XPath 2.0 and we did not know any good criterion of picking a single one, the last 5 queries are all different variants of the Frege style.

As an example, we give the Tarski-style query, U2, copying (4):

```
let $start := doc("mbench.xml")//*[@aLevel=5]
return
$start//*[@aFour=3] except $start//*[not(.[@aFour=0])]//*
```

Note that `//*` is an XPath abbreviation for the `descendant` relation and `except` is the XPath way of writing complementation. Knowing this, the similarity with (4) is clear.

As another example, we show U4, one way of encoding the Frege style in XPath:

```
for $start in doc("mbench.xml")//*[@aLevel=5]
for $end in $start//*[@aFour=3]
return
    $end[every $inbetween in $start//*[.//*[. is $end]]
         satisfies $inbetween[@aFour=0]]
```

This variant uses the `for-return` construct and the universal quantification `every` available in XPath 2.0.

The Kleene style query U1 cannot be expressed in XPath 2.0. Luckily, XQuery supports user-defined recursive functions, and thus we could express the query by replacing the Kleene star by a reference to a user-defined function. In this way, we could still compare it to the other styles.

Results. We ran the queries U1–U7 on the following four XQuery engines:

- SaxonB version 8.6.1 [40]
- Qizx/Open version 1.0 [5]
- MonetDB/XQuery version 0.10 [49], 32 bit compilation.
- Galax version 0.5.0 [22]

MonetDB/XQuery is an full-fledged database management system with an XML/XQuery front-end, while the other engines are stand-alone query processors.

The experiment is run with the help of XCheck[4] [2], a testing platform for XQuery engines. We used an Intel(R) Pentium(R) 4 CPU 3.00GHz, with 2026MB of RAM, running Linux version 2.6.12. For the Java applications (SaxonB and Qizx/Open) 1024MB memory size was allocated. We run each query 2 times and we output the results for the last "warmed-up" run. The times reported are CPU times measuring the complete execution of a query including loading and processing the document and outputting the result.

The results are given in Figure 3. We remark that query execution was stopped when an engine took more than 1000 seconds to output the results. An empty engine bar indicates an engine crash.

The Kleene style query U1 performs best on all of the four engines. The Tarski variant, query U2, does reasonably well on 3 out of 4 engines, with Galax having some troubles, but at least finishing before the timeout limit. The Frege style queries U3, U4, and U5 are really problematic for all the engines. Only SaxonB finishes before the 1000 seconds limit time, Qizx/Open and Galax pass the limit, while MonetDB/XQuery crashes on all three queries. We believe the reason for this bad performance is the large amount of intermediate results generated during the execution. This intermediate results are avoided in the queries U6 and U7, which use the `ancestor` axis for navigating upwards in the tree and checking the until condition B on the paths that start with a starting-node and end on a node satisfying the until condition A.

[4] `http://ilps.science.uva.nl/Resources/XCheck/`

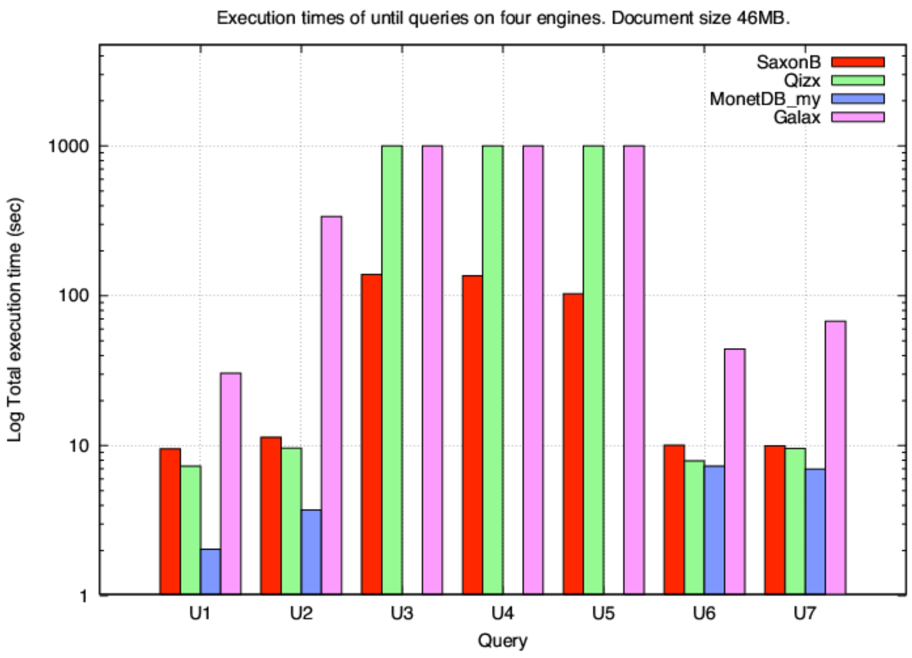


Fig. 3. Processing times of 7 equivalent formulations of the until query for SaxonB, Qizx/Open, MonetDB/XQuery and Galax, on the Michigan benchmark document of 46MB, with a timeout of 1000 seconds

2.7 In Conclusion

The developments around XPath are a nice example of interaction between formal theory and applied computer science. Core XPath 1.0 turned out to be (by accident?) a modal logic, and equivalent to a well defined fragment of first order logic. One of the design goals of XPath 2.0 was to make it as expressive as first-order logic. Regular XPath is an extension of XPath in which Propositional Dynamic Logic and formalisms coming from SGML (caterpillar expressions [11]) and semi-structured data (regular path expressions [1]) nicely blend together.

Hierarchies of formalisms for describing and querying XML-trees form an active topic of research at the moment. Typical questions concern relative expressive power, succinctness, complexity, and various types of translations between different formalisms. There are still many hard nuts to crack (for instance, the question marks in Figure 2).

One potential issue of concern is that we have abstracted away too much from the XML reality by considering XML documents as node-labeled sibling ordered trees. Nodes in XML documents can have attributes with typed values (strings, integers, etc), and atomic data values, and these are neglected in the work we have discussed. One of the reasons is that static analysis problems (like satisfiability) for languages that can speak about these data values quickly become undecidable. Some work has been done in finding well-behaved fragments of XPath with limited access to data values [10], but this is not yet of applicational value. Here input from another field of modal logic is likely to be helpful, namely the work on description logics with concrete domains [45].

3 Logical Foundations of XPath

This section is based on joint work with Balder ten Cate [64].

3.1 Introduction

XPath is a common fragment of the XML querying and processing languages XQuery and XSLT, used for navigation through XML documents. We address two foundational issues concerning XPath: (1) its *expressivity* in comparison to the first-order logic, and (2) *algebras* for XPath.

We focus on the *navigational* part of XPath: the part that is concerned purely with document navigation, not considering operations involving strings, numbers, or any other types of atomic content. Several navigational fragments of XPath 1.0 and 2.0 have been proposed [27,63]. All in all, we consider four navigational XPath dialects: Core XPath 1.0, variable-free Core XPath 2.0, Core XPath 2.0 with variables, and Regular XPath$^{\approx}$.

XML tree navigation using path expressions. Path expressions describe ways of navigating through XML documents, i.e., travelling from one node to another in the tree. This means we can model the meaning of a path expressions by a binary relation on the nodes of the tree. For example, the XPath 1.0 path expression `descendant::p` (abbreviated as `.//p`) denotes in any XML tree T, the set of all pairs (m, n) with n a descendant node of m that has tag name `p`. Of course, binary relations can be defined using many other formalisms, e.g., by means of a first-order formula in two free variables. In the case of this example, the binary relation is equivalently expressed by the conjunctive query

$$\phi(x, y) = \mathtt{descendant}(x, y) \land \mathtt{p}(y) \; .$$

Next we consider a conjunctive query expressing the existence of a path in the tree ending in a q node and having along the way at least z many p_i-nodes (for $1 \leq i \leq z$). Note that the p_i nodes may occur in any order along the path.

$$\phi(x, y) = \exists z_1 \ldots z_n \bigwedge_{i=1}^{n} \mathtt{descendant}(x, z_i) \land \mathtt{p}_i(z_i) \land \mathtt{descendant}(z_i, y) \land \mathtt{q}(y) \tag{6}$$

This binary relation can be defined in XPath 1.0 by the union of the path expressions

$$\mathtt{descendant} :: \mathtt{p}_{\rho(\mathtt{1})} / \cdots / \mathtt{descendant} :: \mathtt{p}_{\rho(\mathtt{n})} / \mathtt{descendant} :: \mathtt{q}$$

for all, exponentially many, permutations ρ of $1 \ldots n$.

Next, consider the following first-order binary relation (familiar from temporal logic, and raising children):

$$\begin{aligned}\phi(x,y) = {} & \texttt{descendant}(x,y) \wedge \mathtt{q}(y) \wedge \\ & \forall z(\texttt{descendant}(x,z) \wedge \texttt{descendant}(z,y) \rightarrow \mathtt{p}(z))\end{aligned} \tag{7}$$

A pair (m, n) stands in this relation if n is a descendant of m with tag name q and all nodes in-between m and n in the tree have tag name p. Can we express this in XPath 1.0?[5]

Questions such as these are hard to answer for languages as rich as full XPath 1.0 (whose technical specification is about 30 pages long). In order to be able to give a mathematically precise answer, in [48] the same question was studied in the context of *Core XPath 1.0* [27]. This is a compact, well defined fragment of XPath 1.0 with a clean logical semantics. It captures the navigational core of XPath 1.0, abstracting away from operations involving strings, numbers, or any other types of atomic content. It was shown in [48] that (7) cannot be defined in Core XPath.

Ways of extending Core XPath 1.0. Various extensions of XPath 1.0 have been proposed, including the official W3C standard of XPath 2.0. With more expressive power, new binary relations can be defined and sometimes older ones can be defined more succinctly. We give examples of both, starting with the latter.

XPath 2.0 has an `intersect` operator: `Path1 intersect Path2` denotes the intersection of the binary relations defined by `Path1` and `Path2`. Using intersection, (6) can be expressed without exponential blow-up:

$$\begin{array}{l}\texttt{descendant} :: \mathtt{p_1}/\texttt{descendant} :: \mathtt{q}\ \texttt{intersect} \\ \texttt{descendant} :: \mathtt{p_2}/\texttt{descendant} :: \mathtt{q}\ \texttt{intersect} \\ \quad\ldots \\ \texttt{descendant} :: \mathtt{p_n}/\texttt{descendant} :: \mathtt{q}\end{array}$$

Similarly, the previously undefinable "until" relation (7) can be defined in various ways using additional operators that have been proposed. A first possibility is to use the Kleene star, inspired by [1]:

$$(\texttt{child} :: \mathtt{p})^*/\texttt{child} :: \mathtt{q}.$$

Here $(\texttt{Path})^*$ denotes the reflexive transitive closure of the binary relation denoted by `Path`. The Kleene star does not belong to XPath 1.0 or 2.0, but extensions of XPath with this operator have been proposed and implemented [61,21,20]. A second solution is to use the *path complementation* operator `except` that was introduced in XPath 2.0:

$$\begin{array}{l}\texttt{descendant} :: \mathtt{q}\ \texttt{except} \\ \quad \texttt{descendant} :: *[\texttt{not}(\texttt{self} :: \mathtt{p})]/\texttt{descendant} :: \mathtt{q}\end{array}$$

[5] Note that $.//\mathtt{q}[\texttt{not}(\texttt{ancestor} :: *[\texttt{not}(\texttt{self} :: \mathtt{p})])]$ does not define the intended relation: it is only correct for pairs (m, n) where m is the root.

Finally, a third option is to use quantified variables, which is possible in XPath 2.0 using the `for`-construct. Using `for`, we can write (7) as follows:

```
for $s in . return
  descendant :: q[not(ancestor :: *[not(self :: p)]/
                      ancestor :: *[. is $s])]
```

Notice how the variable `$s` stores the initial node.

Two main questions. We consider Core XPath 1.0 and three extensions of it, roughly corresponding to XPath 2.0, the variable free fragment of XPath 2.0, and an extension of XPath with transitive closure and path equalities. For each of these, we study two main questions: *what is the expressive power* and *what are suitable algebras*.

Expressivity and Codd completeness. When a new query language is introduced, it is always useful to compare its expressive power to existing languages. E.F. Codd did this for SQL and relational algebra by showing that they are equally expressive as first-order logic [16]. With the navigational languages for XML we can do the same: given a dialect of XPath, we can ask how it compares to (fragments or extensions of) first-order logic. We explore this in Section 3.3.

Algebras for navigational XPath. An important step towards efficient query evaluation is to identify a suitable algebra in which query plans can be formulated. Which algebra are suitable for our XPath dialects? In answering this question, we guide ourselves by the following criteria:

1. expressions in the XPath dialect should be efficiently translatable to algebraic expressions,
2. the algebra should not be much more expressive than the XPath dialect requires,
3. the algebra should not have much harder query evaluation or equivalence problem than the XPath dialect itself, and
4. there should be a nice set of algebraic equivalence rules for the algebra.

In Section 3.4, we will consider several candidates, such as Codd's relational algebra (CRA) and Tarski's algebra of binary relations (TRA). For each dialect of navigational XPath, a different algebra turns out to fit best.

3.2 Preliminaries: Four Dialects of Navigational XPath

In this section, we review the syntax and semantics of Core XPath 1.0 —the navigational fragment of XPath 1.0 introduced in [27]— as well as three extensions.

Core XPath 1.0. Core XPath 1.0 was introduced in [27] to capture the navigational core of XPath 1.0. The definition we will give here is from [47], which differs from the one of [27] as (1) it include the "one-step sibling axes" left, right (which are definable in XPath 1.0 using numerical predicates), (2) filters can be applied to any expression, and (3) we include the union operator on path expressions.

Table 1 gives the syntax of Core XPath 1.0. Here QName stands for any XML tag name. The primary type of expression is a *path expression* (PathExpr).

Table 2 gives the semantics. Expressions are evaluated on finite sibling-ordered unranked trees whose nodes are labeled by XML tag names. Given such a tree, the meaning $[\![R]\!]_{\mathsf{PExpr}}$ of a PathExpr R is always a binary relation. This is just another, equivalent, way of specifying a function from nodes to sets of nodes (the answer-set semantics). The meaning $[\![T]\!]_{\mathsf{NExpr}}$ of a node expression T is always a set of nodes.

We will study the complexity of two tasks: query evaluation and query containment. For *query evaluation*, we will consider the *combined complexity* of the following problem: given a path expression, an XML-tree (suitably encoded) and a pair of nodes, determine whether the pair belongs to the relation denoted by the path expression. In the case of the *query containment* problem, the task is to determine, given two path expressions R, S, whether in every tree model, $[\![R]\!]_{\mathsf{PExpr}} \subseteq [\![S]\!]_{\mathsf{PExpr}}$. For Core XPath 1.0, the query evaluation problem for Core XPath 1.0 is in PTIME (in fact, it can be performed in linear time) [27], and the query containment problem is EXPTIME-complete [46,50].

Core XPath 2.0 without variables. In [63], Core XPath 2.0 was introduced as a navigational core of XPath 2.0 with a clean, logical semantics. One important simplifying assumption underlies Core XPath 2.0, namely that path expressions still denote *binary relations between nodes*, as they did in Core XPath 1.0. This is not the case in the full XPath 2.0, where they denote functions from nodes to sequences of nodes (not

Table 1. Syntax of Core XPath 1.0

Axis	:=	`self` | `child` | `parent` | right | left
	|	`descendant`
	|	`ancestor`
	|	`following`
	|	`preceding`
	|	`following_sibling`
	|	`preceding_sibling`
NameTest	:=	QName | $*$
Step	:=	Axis::NameTest .
PathExpr	:=	Step
	|	PathExpr/PathExpr
	|	PathExpr `union` PathExpr
	|	PathExpr[NodeExpr]
NodeExpr	:=	PathExpr
	|	`not` NodeExpr
	|	NodeExpr `and` NodeExpr
	|	NodeExpr `or` NodeExpr.

Table 2. Semantics of Core XPath 1.0

$$
\begin{array}{ll}
[\![\mathsf{Axis} :: N]\!]_{\mathsf{PExpr}} & = \{(x,y) \mid x\mathsf{Axis}y \text{ holds in the tree, and } y \text{ has tag } N\} \\
[\![\mathsf{Axis} :: *]\!]_{\mathsf{PExpr}} & = \{(x,y) \mid x\mathsf{Axis}y \text{ holds in the tree}\} \\
[\![R/S]\!]_{\mathsf{PExpr}} & = [\![R]\!]_{\mathsf{PExpr}} \circ [\![S]\!]_{\mathsf{PExpr}} \\
[\![R \texttt{ union } S]\!]_{\mathsf{PExpr}} & = [\![R]\!]_{\mathsf{PExpr}} \cup [\![S]\!]_{\mathsf{PExpr}} \\
[\![R[T]]\!]_{\mathsf{PExpr}} & = \{(x,y) \mid (x,y) \in [\![R]\!]_{\mathsf{PExpr}} \text{ and } y \in [\![\mathrm{T}]\!]_{\mathsf{NExpr}}\} \\
\\
[\![\mathsf{PathExpr}]\!]_{\mathsf{NExpr}} & = \{x \mid \exists y.(x,y) \in [\![\mathsf{PathExpr}]\!]_{\mathsf{PExpr}}\} \\
[\![\texttt{not } T]\!]_{\mathsf{NExpr}} & = \{x \mid x \notin [\![T]\!]_{\mathsf{NExpr}}\} \\
[\![T_1 \texttt{ and } T_2]\!]_{\mathsf{NExpr}} & = [\![T_1]\!]_{\mathsf{NExpr}} \cap [\![T_2]\!]_{\mathsf{NExpr}} \\
[\![T_1 \texttt{ or } T_2]\!]_{\mathsf{NExpr}} & = [\![T_1]\!]_{\mathsf{NExpr}} \cup [\![T_2]\!]_{\mathsf{NExpr}}
\end{array}
$$

necessarily in document order and possibly containing duplicates). We follow the definition of Core XPath 2.0 from [63].

First, we consider the variable-free fragment of Core XPath 2.0. This is a very simple extension of Core XPath 1.0: it differs from Core XPath 1.0 only in that one can take intersections and complements of path expressions:

$$
\begin{array}{ll}
[\![R \texttt{ intersect } S]\!]_{\mathsf{PExpr}} & = [\![R]\!]_{\mathsf{PExpr}} \cap [\![S]\!]_{\mathsf{PExpr}} \\
[\![R \texttt{ except } S]\!]_{\mathsf{PExpr}} & = [\![R]\!]_{\mathsf{PExpr}} \setminus [\![S]\!]_{\mathsf{PExpr}}.
\end{array}
$$

These operators do not only increase the expressive power of the language (as we will see in the next section), they also greatly increase its complexity. The query evaluation problem for variable free Core XPath 2.0 is still in PTIME (in fact, it can be performed in quadratic time), but the query containment problem is non-elementary (2-EXPTIME-complete for expressions without the complementation operator) [62].

Core XPath 2.0. Besides the addition of the `intersect` and `except` operators, an important difference between XPath 1.0 and 2.0 is the use of quantified variables by means of the `for` construct. Formally, let a NodeRef expression be an expression of the form `$i` or . (where `$i` is a variable ranging over nodes in the tree). Then the syntax of full Core XPath 2.0 is obtained by extending the syntax of Core XPath 1.0 with the `intersect` and `except` operators from above, with path expressions of the form `$i` and `for $i in` PathExpr `return` PathExpr, and with node expressions of the form NodeRef `is` NodeRef. The latter tests whether the two expressions refer to the same node.

Since the expressions of Core XPath 2.0 can contain variables, the semantic interpretation is relative to an *assignment*, i.e., a function mapping variables to nodes. For g an assignment, `$i` a variable, and x a node, $g[\$\mathtt{i} \mapsto x]$ denotes the assignment g' which is identical to g except that $g'(i) = x$. Also, for any assignment g, node x, and NodeRef expression a, let $[\![a]\!]^{g,x}$ be $g(a)$ in case a is a variable, or x in case a is '`.`'. The semantics of the new constructs is as follows:

$$[\![\texttt{\$i}]\!]^g_{\mathsf{PExpr}} = \{(x, y) \mid g(i) = y\}$$

$$[\![\texttt{for}\,\texttt{\$i}\,\texttt{in}\,R\,\texttt{return}\,S]\!]^g_{\mathsf{PExpr}} =$$
$$\{(x, y) \mid \exists z((x, z) \in [\![R]\!]^g_{\mathsf{PExpr}} \text{ and } (x, y) \in [\![S]\!]^{g[\$i\mapsto z]}_{\mathsf{PExpr}})\}$$

$$[\![a \texttt{ is } b]\!]_{\mathsf{NExpr}} = \{x \mid [\![a]\!]^{g,x} = [\![b]\!]^{g,x}\}.$$

The query evaluation problem for Core XPath 2.0 is PSPACE-complete, and the query containment problem is non-elementary [62].

Regular XPath$^{\approx}$. Regular XPath$^{\approx}$ extends Core XPath 1.0 with two operators that are not part of XPath 1.0 or 2.0, and that, as we will see, make it more expressive. The most important of these is the Kleene star, which allows us to take the reflexive transitive closure of arbitrary path expressions. The other is *path equalities* (not to be confused with data value equalities). Formally, the semantics of these operators is as follows [61]:

$$[\![R^*]\!]_{\mathsf{PExpr}} = \text{reflexive transitive closure of } [\![R]\!]_{\mathsf{PExpr}}$$

$$[\![R \approx S]\!]_{\mathsf{NExpr}} = \{x \mid \exists y.(x, y) \in [\![R]\!]_{\mathsf{PExpr}} \cap [\![S]\!]_{\mathsf{PExpr}}\}$$

Regular XPath$^{\approx}$ can be viewed as a mix between Core XPath 1.0 and *regular path expressions* [1]: it has the filter expressions of the former and the Kleene star of the latter. It is still mainly studied in the theoretical community [21,26,61].

The query evaluation problem for Regular XPath$^{\approx}$ is in PTIME (in fact, in quadratic time), and the query containment problem is EXPTIME-complete [62].

3.3 Expressivity of XPath Dialects

We have defined four XPath fragments. How do they compare in terms of expressivity and succinctness? We will answer this question by mapping each XPath dialect to an equally expressive variant of first-order logic.

Since the data model of an XML document is a finite sibling ordered tree, it is natural to consider first-order logic in the signature with eight atomic binary relations corresponding to the basic axes (`child`, `parent`, `left` and `right`, and their transitive closures `descendant`, `ancestor`, `following-sibling` and `preceding-sibling`) plus a unary predicate for each tag name. We will call the first order language in this signature FO^{tree}. With $FO^{\mathsf{tree}}(x)$ and $FO^{\mathsf{tree}}(x, y)$ we denote the FO^{tree} formulas in one and two free variables, respectively.

Besides looking at expressive power, we will also compare different languages in terms of *succinctness*. As usual, if two languages, L and L', are equally expressive, we say that L is *(at least) exponentially more succinct than* L' if there is a infinite sequence of L-expressions $R_1, R_2, \ldots$ where the length of R_k is polynomial in k, such that for every sequence of equivalent L'-expressions $R'_1, R'_2, \ldots$, the length of R'_k is exponential in k. Similarly, one can say that a language is *non-elementarily more succinct* than another language.

The results from this section are summarized in Table 3. These results hold both for path expressions and for node expressions.

Table 3. Expressivity and succinctness of XPath dialects

XPath dialect	Core XPath 1.0	$\subsetneq$	Variable-free Core XPath 2.0	$\equiv$	Core XPath 2.0	$\subsetneq$	Regular XPath$^{\approx}$
Equivalent FO-dialect	$\exists FO^{\mathsf{tree}^{\mathrm{mon}\neg}}$		FO^{tree}		FO^{tree}		$FO^{\mathsf{tree}*}$
	(exponential succinctness gap)		(at least exponential succinctness gap)		(no succinctness gap: linear translations)		(non-elementary succinctness gap)

The results discussed in this section naturally build on a existing line of research in temporal logic, which originates in the work of H. Kamp [36] and which studies expressive completeness for various temporal logics on trees. A survey of this area may be found in [34].

Core XPath 1.0. As we have already seen in Section 3.1, not every FO^{tree}-definable binary relation is definable in Core XPath 1.0. However, we can define a natural fragment of FO^{tree} with respect to which Core XPath 1.0 is complete.

Let $\exists FO^{\mathsf{tree}^{(\mathrm{mon}\neg)}}$ be the fragment of FO^{tree} where negation can only be applied to subformulas with exactly one free variable, and universal quantification is disallowed altogether (thus, the connectives are conjunction, disjunction, and existential quantification, plus negation of formulas with at most one free variable). It can be seen from Table 2 that Core XPath 1.0 path expressions can be translated into this fragment of FO^{tree} (indeed, the only form of negation present in Core XPath 1.0 is negation in filter expressions, which corresponds to negation of a formula in one free variable). A converse translation is possible as well, although it involves an exponential blow-up (recall the example we gave in the introduction):

Theorem 1. (Core XPath 1.0 $\equiv \exists FO^{\mathsf{tree}^{(\mathbf{mon}\neg)}}(x, y)$)

1. *There is a linear translation from Core XPath 1.0 path expressions to $\exists FO^{\mathsf{tree}^{(mon\neg)}}(x, y)$ formulas, and an exponential translation backwards.*
2. *Indeed, $\exists FO^{\mathsf{tree}^{(mon\neg)}}(x, y)$ formulas are exponentially more succinct than Core XPath 1.0 path expressions.*

Proof. The difficult direction of (1) can be proved by induction on the nesting depth of negation, using the fact that positive existential fist-order formulas can be translated to Core XPath 1.0 path expressions at the cost of an exponential blowup [7,28]. For the exponential difference in succinctness, see [62, Thm. 26].

An alternative characterization of Core XPath 1.0, in terms of conjunctive queries and the two-variable fragment of FO^{tree}, is given in [48].

Core XPath 2.0. In the case of Core XPath 2.0, there is a precise match with FO^{tree}, in terms of expressive power. In fact, this Codd-completeness has been one of the design considerations for XPath 2.0 [38]. Moreover, it turns out to hold already for the variable free fragment. Still, the presence of variables matters for the succinctness of the language.

For simplicity, we consider only path expressions that have no *free variables*. For a discussion of expressive completeness in the presence of free variables, see [23].

Theorem 2. (Core XPath 2.0 $\equiv FO^{\mathsf{tree}}(x, y)$**)**

1. *There are linear translations between Core XPath 2.0 path expressions and* $FO^{\mathsf{tree}}(x, y)$ *formulas.*
2. *There is a linear translation from variable free Core XPath 2.0 path expressions to* $FO^{\mathsf{tree}}(x, y)$ *formulas and a non-elementary translation backwards.*
3. $FO^{\mathsf{tree}}(x, y)$ *formulas are at least exponentially more succinct than variable free Core XPath 2.0 path expressions.*

Proof. The linear translations are straightforward. A non-elementary translation from FO^{tree} to variable free Core XPath 2.0 is given in [47]. The exponential difference in succinctness between FO^{tree} and variable free Core XPath 2.0 holds already on linear orders (i.e., documents in which each node has at most one child) [29].

In fact, it was shown in [47] that a more modest extension of Core XPath 1.0 called *Conditional XPath* is already expressively complete for FO^{tree}. It extends Core XPath 1.0 with "conditional axes" of the form $(\mathsf{Axis}\ \mathtt{while}\ \mathsf{NodeExpr})$, with $\mathsf{Axis} \in \{\mathtt{child}, \mathtt{parent}, \mathtt{left}, \mathtt{right}\}$. Without going into further details, we only mention that $(\mathsf{Axis}\ \mathtt{while}\ T)\ ::\ N$ can be written in Core XPath 2.0 as

$$\mathsf{Axis}^{+} :: N\ \mathtt{except}\ (\mathsf{Axis}^{+} :: *[\mathtt{not}(T)]/\mathsf{Axis}^{+} :: *)$$

where Axis^{+} is the transitive version of Axis.

Regular XPath$^{\approx}$. Since the conditional axes of [47] are definable in Regular XPath$^{\approx}$ using the Kleene star — $(\mathsf{Axis}\ \mathtt{while}\ T)::N$ is equivalent to $(\mathsf{Axis}::*[\mathtt{not}(T)])^{*}/\mathsf{Axis}::N$ — we already know by [47] that Regular XPath$^{\approx}$ extends FO^{tree} in expressive power. In order to give a precise characterization of the expressive power of Regular XPath$^{\approx}$, we must consider an extension of FO^{tree}.

The simplest option is to simply extend FO^{tree} with a Kleene star (i.e., a transitive closure operator for binary relations). Thus, let $FO^{\mathsf{tree}^*}(x, y)$ be the extension of FO^{tree} (x, y) with a transitive closure operator that applies to formulas with exactly two free variables. Then the following is proved in [61] and [62, Thm. 27]:

Theorem 3. (Regular XPath$^{\approx}$ $\equiv FO^{\mathsf{tree}^*}$**)**

1. *There is a linear translation from Regular XPath$^{\approx}$ path expressions to* $FO^{\mathsf{tree}^*}(x, y)$ *formulas, and a non-elementary translation backwards.*
2. *In fact,* $FO^{\mathsf{tree}^*}(x, y)$ *formulas are non-elementarily more succinct than Regular XPath$^{\approx}$ path expressions.*

Incidentally, FO^{tree^*} is not the same as $FO^{\mathsf{tree}} + TC^1$: the standard unary transitive closure operator TC^1 can be applied to formulas containing more than two free variables, as long as two of the variables are designated; the others are treated as parameters (cf. for instance [18]). We do not know at present whether FO^{tree^*} and $FO^{\mathsf{tree}} + TC^1$ have the same expressive power on trees. The following question is also open at the time of writing: is Regular XPath without $\approx$ equally expressive as $FO^{\mathsf{tree}^*}(x, y)$?

3.4 Algebras for XPath Dialects

The previous section discussed that Core XPath 2.0 corresponds in expressive power to exactly first-order logic. The next question is which algebras are appropriate for representing query plans for Core XPath 2.0 expressions. The same question holds for the other dialects we discussed. Codd's relational algebra seems a natural choice because it is again equally expressive as first-order logic. Indeed, we will see that it is a good choice when considering Core XPath 2.0. For other XPath dialects however (including the variable free fragment of Core XPath 2.0), there are better options.

In Section 3.1 we gave criteria for determining whether an algebra is suitable for an XPath dialect. In this section, we discuss four different algebras, and determine which ones match best with each XPath dialect. The results are summarized in Table 5.

To simplify the presentation, we will first consider Core XPath 1.0 and 2.0, and only afterwards Regular XPath$^{\approx}$, as the latter requires (a mild form of) recursion in the algebra.

Four candidate algebras

Codd's relational algebra (CRA) and its fragment CRA($mon\neg$). We briefly recall Codd's relational algebra. A characteristic feature of this algebra is that it is *many sorted*: each expression has an associated *arity* corresponding to the number of columns of the table it computes. The atomic expressions are simply the names of the relations in the database, and the operations are *selection* (σ), *projection* (π), *cross-product* ($\times$), *union* ($\cup$) and *complementation* ($-$).

The fact that there is no bound on the arity of the expressions has some negative consequences on the complexity of query evaluation: it is PSPACE-complete, whereas it becomes polynomial if there is a bound on the allowed arity of (sub)expressions [12,67].

Inspired by the results in the previous section, it makes sense to distinguish another restricted fragment of CRA, namely $CRA(mon\neg)$. This fragment is obtained by restricting the use of complementation to unary tables. Note that all $SPCU$-expressions still belong to this fragment.

Tarski's relation algebra (TRA). Tarski's relation algebra [59,60] is an algebra of binary relations: each expression denotes a table with precisely two columns. The operations on binary relations considered by Tarski are the Boolean operations (union, intersection and complementation), as well as composition $\circ$ and converse $(\cdot)^{-1}$. There are also two constants (or, 0-ary operations) $\top$ and ϵ, which stand for the total relation and the identity relation (over the given domain). A typical example of an equivalence in this algebra is $\alpha \circ (\beta \cup \gamma) \equiv \alpha \circ \beta \cup \alpha \circ \gamma$.

It was shown in [60] that TRA has the same expressive power as the three-variable fragment of first-order logic in two free variables, over vocabularies consisting of binary relations only.

Although in TRA all expressions denote binary relations, unary relations can be easily dealt with as well, for instance by treating them as subrelations of the identity relation (e.g., $\{a, b, c\}$ can be treated as $\{(a, a), (b, b), (c, c)\}$).

Table 4. Complexity of evaluation and containment for the algebras on trees

	Evaluation	*Containment*
CRA	PSPACE-compl.	Non-elementary
$CRA(mon\neg)$	NP-hard, in P^{NP}	2-EXPTIME-compl.
TRA	PTIME (quadratic)	Non-elementary
DRA	PTIME (linear)	EXPTIME-compl.

Dynamic relation algebra (DRA). In [35,66], a reduct of Tarski's relation algebra is studied containing only the operations $\cup$, $\circ$ and $\sim$. The latter of these is called the *counterdomain* operation. It takes a binary relation R and produces a subrelation of the identity relation: $\sim R$ denotes $\{(x, y) \mid x = y$ and $\neg\exists z.(x, z) \in R\}$. In TRA, it can be expressed as $\epsilon - (R \circ \top)$. This operator is quite handy: e.g., $\sim$`child` expresses "I am a leaf node", and $\sim\sim$`child` expresses "I am *not* a leaf node". We call this algebra *dynamic relation algebra* (DRA).

The signature of DRA might seem poor, but it is rich enough to capture all of Core XPath 1.0. If we encode properties of nodes as subrelations of the identity relation (as we already suggested above), then we have the following translation:

$$
\begin{aligned}
&\mathsf{TR}_{\mathsf{PExpr}}(\mathsf{Axis} :: *) &&= \mathsf{Axis}\\
&\mathsf{TR}_{\mathsf{PExpr}}(\mathsf{Axis} :: N) &&= \mathsf{Axis} \circ N\\
&\mathsf{TR}_{\mathsf{PExpr}}(R/S) &&= \mathsf{TR}_{\mathsf{PExpr}}(R) \circ \mathsf{TR}_{\mathsf{PExpr}}(S)\\
&\mathsf{TR}_{\mathsf{PExpr}}(R \ \texttt{union} \ S) &&= \mathsf{TR}_{\mathsf{PExpr}}(R) \cup \mathsf{TR}_{\mathsf{PExpr}}(S)\\
&\mathsf{TR}_{\mathsf{PExpr}}(R[T]) &&= \mathsf{TR}_{\mathsf{PExpr}}(R) \circ \mathsf{TR}_{\mathsf{NExpr}}(T)\\[4pt]
&\mathsf{TR}_{\mathsf{NExpr}}(\mathsf{PathExpr}) &&= \sim\sim \mathsf{TR}_{\mathsf{PExpr}}(\mathsf{PathExpr})\\
&\mathsf{TR}_{\mathsf{NExpr}}(\texttt{not} \ T) &&= \sim \mathsf{TR}_{\mathsf{NExpr}}(T)\\
&\mathsf{TR}_{\mathsf{NExpr}}(T_1 \ \texttt{and} \ T_2) &&= \mathsf{TR}_{\mathsf{NExpr}}(T_1) \circ \mathsf{TR}_{\mathsf{NExpr}}(T_2)\\
&\mathsf{TR}_{\mathsf{NExpr}}(T_1 \ \texttt{or} \ T_2) &&= \mathsf{TR}_{\mathsf{NExpr}}(T_1) \cup \mathsf{TR}_{\mathsf{NExpr}}(T_2)
\end{aligned}
$$

In [66], an elegant model theoretic characterization of DRA is given in terms of *safety for bisimulations*.

DRA is a fragment of both TRA and CRA$(mon\neg)$. More precisely, the relationships between the four algebras on arbitrary models are as follows:

$$\begin{tikzcd}
 & \text{TRA} \arrow[dr, phantom, sloped, "\subsetneq"] & \\
\text{DRA} \arrow[ur, phantom, sloped, "\subsetneq"] \arrow[dr, phantom, sloped, "\subsetneq"] & & \text{CRA} \\
 & \text{CRA}(mon\neg) \arrow[ur, phantom, sloped, "\subsetneq"] &
\end{tikzcd}$$

When we restrict attention to XML documents (i.e., where the atomic relations are the 8 binary relations corresponding to the different axes, as well a "unary" relation for each of the different tag names), the situation is a bit different: on this restricted class of models TRA and CRA have the same expressive power, as do DRA and CRA$(mon\neg)$.

Complexity of these algebras on trees. We will now discuss the complexity of *query evaluation* and *query containment* for the four algebras interpreted on XML-trees (i.e., where the atomic relations are the 8 binary relations corresponding to the different axes, as well a "unary" relation for each of the different tag names). As before, in the case of query evaluation we consider the combined complexity of testing whether a given pair belongs to the relation defined by a given path expression on a given XML document. Table 4 provides a summary of the results.

Containment. By Rabin's theorem, query containment is decidable for all four algebras. For TRA and CRA, containment is non-elementary, as follows from Stockmeyer's non-elementary lower bound for the non-emptiness problem of star-free expressions [58,62]. The results for CRA($mon\neg$) and DRA follow from known results about XPath. In particular, the 2-EXPTIME-hardness of CRA($mon\neg$) query containment follows from the same lower bound for Core XPath 1.0 extended with path intersection, as the latter can be linearly translated into CRA($mon\neg$). The upper bound follows from the existence of a singly exponential translation from CRA($mon\neg$)-expressions of arity 2 to Core XPath 1.0, and the fact that Core XPath 1.0 has an EXPTIME-complete query containment problem (the restriction to expressions of arity 2 is not essential: containment of CRA($mon\neg$)-expressions of arity greater than 2 can be linearly reduced to containment of ones of arity 2, in fact to Boolean CRA($mon\neg$)-expressions) [62]. The result for DRA follows from linear translations to Core XPath 1.0.

Evaluation. The combined complexity of query evaluation for CRA is PSPACE-complete, also when restricted to XML-trees [12]. As TRA corresponds to a fixed variable fragment of first-order logic the complexity drops to PTIME. Using the bottom-up algorithm sketched in [67] it can be shown to be in $O(n^2)$. In [27], it is shown that query evaluation for Core XPath 1.0 can be performed in linear time. Because Core XPath 1.0 and DRA linearly translate to each other, the result transfers to DRA. Recall that we are not talking about the complexity of computing the relation denoted by a path expression (which could be quadratic in the size of the tree), but of the complexity of checking whether a given pair of nodes belongs to the denotation of a given expression in a given tree. Query evaluation for CRA($mon\neg$) is NP-hard: this holds even for positive conjunctive queries with only downward axis relations [28]. For the P^{NP}-upper-bound, we use an algorithm that runs in polynomial time and that uses an oracle for testing whether a tuple belongs to the answer set of an SPCU-expression. The algorithm proceeds roughly as follows: given an expression α, it starts by listing all subexpressions whose main connective is a (unary) complementation operator, in order of growing length. One by one, it computes for each such subexpression α the (polynomially large) answer set, by asking the oracle for each element whether it belongs to the answer set. The occurrences of α within larger expressions are then replaced by the computed answer set. Finally, we are left with a single SPCU-expression, to which the oracle is once more applied.

Axiomatizations. One of our criteria for being a good algebra was the availability of an axiomatization of the valid equations on XML-trees (finite sibling ordered node-labeled trees). Only a few results are known here. In [7], an axiomatization is given for the $\sim$-free reduct of Dynamic Relation Algebras DRA with only the two downward axis plus atomic label tests. An axiomatization of the full DRA on XML-trees is not known (a complete axiomatization on arbitrary models is given in [35]). In [63], an axiomatization of first-order logic on XML-trees is given, from which an axiomatization for TRA on XML-trees is derived. We believe that in a similar way an axiomatization of CRA on XML-trees can be found. The TRA axiomatization consists of general axioms for the TRA similarity type like $R \circ (S \circ T) = (R \circ S) \circ T$ plus special axioms which are only valid on trees. Two examples are Tr5 and Tr11:

$$\begin{array}{ll} \text{Tr5.} & \downarrow^+ \circ \uparrow^+ \quad \equiv \quad \downarrow^+ [\downarrow] \cup \epsilon[\downarrow] \cup (\epsilon[\downarrow] \circ \uparrow^+) \\ \text{Tr11.} & \epsilon \cup \uparrow^+ \cup \downarrow^+ \cup \\ & (\uparrow^* \circ \rightarrow^+ \circ \downarrow^*) \cup (\uparrow^* \circ \leftarrow^+ \circ \downarrow^*) \quad \equiv \quad \top \end{array}$$

Here we abbreviate the steps in the trees by arrows, e.g., $\downarrow$ is the `child` axis, $\uparrow$ is `parent`, etc. E.g., in XPath notation, the left-hand side of Tr5 would be `descendant/ancestor`. Also, we use $R[S]$ as a shorthand for $R \circ \sim\sim S$. Tr5 is a natural complexity reducing equivalence when read from left to right. Tr11 states the well known fact that the self, ancestor, descendant, following and preceding axis relations partition each XML-tree from every given node.

Which algebra for which XPath? We now have three XPath dialects (Regular XPath$^{\approx}$ will be dealt with in the next subsection) and four candidate algebras. We determine which algebra fits best to which fragment by answering the following questions, corresponding to the first three requirements from Section 3.1:

1. Is there a linear translation from the expressions in XPath dialect to expressions in the algebra?
2. Are the XPath dialect and the algebra equally expressive?
3. Do the XPath dialect and the algebra have the same query containment and evaluation complexities?

(as for the fourth requirement, concerning the existence of nice sets of algebraic equivalence rules for the algebra, we have too little information at present to say much about it).

The answers, based on the results discussed in the previous sections, are given in Table 5. The combinations with only affirmative answers are marked by a gray background.

Regular XPath$^{\approx}$. For Regular XPath$^{\approx}$, the algebras need to be extended with a transitive closure operator. In the case of TRA and DRA, the semantics of such an operator is clear: the denotation of R^* is the reflexive, transitive closure of the binary relation denoted by R. In the case of CRA and CRA($mon\neg$) a similar proviso needs to be made as for FO^{tree^*} (cf. Section 3.3): the transitive closure operator may only be applied to expressions that denote tables with precisely two columns. We use CRA($*$), CRA($mon\neg, *$), TRA($*$) and DRA($*$) to denote the extensions of the respective algebras with the transitive closure operator, which conform to this restriction.

Table 5. Which algebra for which XPath dialect?

	CRA	CRA($mon\neg$)	TRA	DRA
Core XPath 1.0	**Y** (linear translation) **N** (too expressive) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **N** (too expressive) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)
Core XPath 2.0 w/o variables	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**N** (no translation possible) **N** (too little expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)
Core XPath 2.0 with variables	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)	**N** (no elem. translation) **Y** (same expressivity) **Y** (same complexity)	**N** (no translation possible) **N** (too little expressivity) **Y** (lower complexity)
	CRA($*$)	CRA($mon\neg, *$)	TRA($*$)	DRA($*$, loop)
Regular XPath$^{\approx}$	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **N** (complexity too high)	**Y** (linear translation) **Y** (same expressivity) **Y** (same complexity)

The path equalities of Regular XPath$^{\approx}$ can be expressed in CRA($*$) and CRA($mon\neg, *$) using intersection and projection, and in TRA($*$) using intersection and $\sim$: $R \approx S$ can be expressed as $\sim\sim (R \cap S)$. On the other hand, in DRA($*$) it is not clear whether path equalities can be expressed. Let DRA($*$,loop) denote the extension of DRA with both the Kleene star and the $(\cdot)^{\mathsf{loop}}$ operator, that has the following semantics [26]: $R^{\mathsf{loop}} = R \cap \epsilon$. Using loop, and given the fact that Regular XPath$^{\approx}$ is closed under taking inverses of path expressions, we can express path equalities: $R \approx S$ translates to $(R \circ S^{-1})^{\mathsf{loop}}$.

It follows from Theorem 3 that Regular XPath$^{\approx}$, DRA($*$,loop), TRA($*$), CRA($*$) and CRA($mon\neg, *$) all have the same expressive power. Of these four, DRA(*,loop) is the most suitable algebra for Regular XPath$^{\approx}$, since its query containment problem is ExpTime-complete (as follows from the fact that there are linear translations from and to Regular XPath$^{\approx}$ [62]). See also Table 5.

4 Case Study in Making Structure Explicit: Parliamentary Proceedings

This section is based on joint work with Tim Gielissen [25].

4.1 Introduction

Parliamentary proceedings are an interesting set of data to apply state-of-the-art information retrieval technology. Parliamentary proceedings are written records of parliamentary activities containing a wide range of document types. Parliamentary debates are highly structured transcripts of meetings of politicians in parliament. These debates are an important part of the cultural heritage of countries; they are often free of copyright; citizens often have a legal right to inspect them; and several countries make great effort to digitize their entire historical collection and open that up to the general public. This provides many opportunities for the IR (Information Retrieval) community.

We analyze the structure of the parliamentary proceedings and sketch a widely applicable DTD (Document Type Definition, see `http://www.w3schools.com/DTD`). We show how proceedings in PDF format can be transformed into deeply nested XML. We call this process "exemelification". Having the proceedings in XML makes a wide range of applications possible. We elaborate on four of these: entry point retrieval, advanced content and structure search; automatic creation of tables of contents and hyperlinked navigation menus; large savings on storage space and bandwidth for scanned documents.

We only discuss notes of meetings of parliament. As with all meeting notes, these records have the purpose to store the content of the meeting. They have varying degrees of detail. Currently in most Western democracies it is common to transcribe everything that is being said, keeping the content, but making it grammatically correct and pleasant to read.

We list a number of characteristics which make these documents of special interest to the IR community:

- large historical corpora; For example, in Holland all data from 1814 will be available in 2010, at the time of writing it is available since 1974; for the Flemish parliament all data since 1971 is available in PDF; the British Hansard archives have all parliamentary minutes since 1803 available in XML.
- documents contain a lot of consistently applied structure which is rather easy to extract and make explicit;
- transcripts of meetings might be accompanied by audio and video recordings, creating interconnected multimedia data [56];
- data integration issues and opportunities [31,43,44] both within one country (collections from different periods, in different formats, styles, language, . . .), and across countries (cross-lingual IR);
- natural corpus for content and structure queries, combining keyword search with XPath navigation and selection [37,51];
- natural corpus for search tasks in which the answers do not consist of documents: *expert* or *people search* [6], video search[6] and *entry point retrieval* [57].

From this list, we treat the information extraction, data integration and entry-point retrieval aspects. The section is organized as follows: Section 4.2 describes the structure of parliamentary meetings and formalizes it in a DTD. Section 4.3 describes the techniques used in the exemelification process. We discuss four benefits of exemelified data in Section 4.4 and conclude in Section 4.5.

A search engine containing all Dutch parliamentary data from 1984 till May 2008 is built and can be used at `http://www.polidocs.nl`. The corpus of over 80.000 XML files is available for research on request.

4.2 Structure of Parliamentary Proceedings

Notes of a formal meeting with an agenda (e.g., business meeting, council meeting, meeting of the members of a club, etc) are full of implicit structure and contain many common elements. The notes of meetings with a large historical tradition, like parliamentary debates, are in a uniform format which fluctuates little over time. This makes these notes well suited for text-mining.

To our knowledge there is at the time of writing no DTD or markup language for meeting notes available[7].

Transcripts of a meeting contain three main structural elements:

the topics discussed in the meeting (the agenda);

the speeches made at the meeting: every word that is being said is recorded together with 1) the name of the speaker, 2) her affiliation and 3) in which role or function the person was speaking;

non verbal content or actions These can be:

- list of present and absent members;

[6] As done in the TRECVID workshop: `http://www-nlpir.nist.gov/projects/trecvid/`

[7] The DTD of the XML versions of the British Hansard is effectively just a container to store the text, and not suitable as a genuine model of meeting notes.

- description of actions like *applause by members of the Green Party*;
- description of the outcome of a vote;
- the attribution of reference numbers to actions or topics;
- and much more.

The analogy with the structural elements in theatrical drama is striking: scenes, speeches and stage-directions are the theatrical counterparts of the three elements just listed. These are prominent elements in the XML version of Shakespeare's work[8]. The close relation between politics and drama is an emerging theme in political science, see e.g., [30,33].

These elements are structured as follows[9]:

$$\begin{array}{lll}
\texttt{meeting} & \longrightarrow & \texttt{(topic)+} \\
\texttt{topic} & \longrightarrow & \texttt{(speech} \mid \texttt{stage-direction)+} \\
\texttt{speech} & \longrightarrow & \texttt{(p} \mid \texttt{stage-direction)+} \\
\texttt{p} & \longrightarrow & \texttt{(\#PCDATA} \mid \texttt{stage-direction)*} \\
\texttt{stage-direction} & \longrightarrow & \texttt{(\#PCDATA).}
\end{array}$$

All elements contain metadata stored in attributes. The British digitized debates from 1803 till 2004 are available in XML[10] and basically have this structure[11].

Within the Dutch proceedings however there is an intermediate structural element — the block— which distinguishes the theatre drama from the political debate. In Dutch parliament, the debate on each topic is organized as follows: each party may hold a speech by a member standing at the central lectern; other members may interrupt this speech; the chairman can always interrupt everyone. Most often, when all parties had their say at the central lectern, a member of government answers all raised concerns while speaking from the government table and again he or she can be interrupted. In most cases this concludes a topic, but variations are possible and occur (e.g., several members of government speaking or a second round of the whole process).

The *block* is an important debate-structural element because it indicates who is being attacked by the interrupters. Thus for the Dutch situation the DTD becomes

$$\begin{array}{lll}
\texttt{topic} & \longrightarrow & \texttt{(block)+} \\
\texttt{block} & \longrightarrow & \texttt{(speech} \mid \texttt{stage-direction)+}
\end{array}$$

[8] `http://metalab.unc.edu/bosak/xml/eg/shaks200.zip` One of the referees pointed out the well-documented DTD for drama which is part of the TEI guidelines for text markup (`http://www.tei-c.org/release/doc/tei-p5-doc/en/html/DR.html`). This DTD is a good starting point for modelling, but for our purposes both too general and too specific.

[9] PCDATA means *Parsed Character Data*. PCDATA is text that will be parsed by a parser. The text will be examined by the parser for entities and markup.

[10] `http://www.hansard-archive.parliament.uk/`

[11] The structure though is flat instead of nested as it is here, which makes retrieval quite cumbersome. For instance, to retrieve all text spoken by MP `X` we must collect all following siblings of the `member` element which contains the name `X` which come before the next `member` element. We note that this is an example of an until-like query which is not expressible in Core XPath 1.0 [48].

If this block structure is not present in meeting notes, then each topic will have exactly one block child. Thus both types of meeting fit this DTD.

Note. For presentation purposes, the DTD presented here is the core of the model. The DTD actually used contains additional elements and attributes for storing all kinds of metadata. Untill now, DTD is expressive enough for the structure that we want to capture. But we need the possibility of XML Schema to constrain data-types like dates.

Figure 4 contains a visualization of a one-topic debate which uses the block structure and which is created with an XSL-stylesheet from the XML. Each row stands for one block and each vertically positioned mouth stands for one speech. The size of the mouth is proportional to the length of the speech measured in number of words. The speaker on the central lectern has the red mouth, the interrupters have a blue mouth. Interruptions by the chairman are not shown.

We end this section with two more observations on interesting structure in debates, also visible in Figure 4:

1. Blocks consist either of one uninterrupted speech or they have the form `(red,blue)+,red`, that is a sequence of pairs of speeches by the central speaker and an interrupter ended by the central speaker.

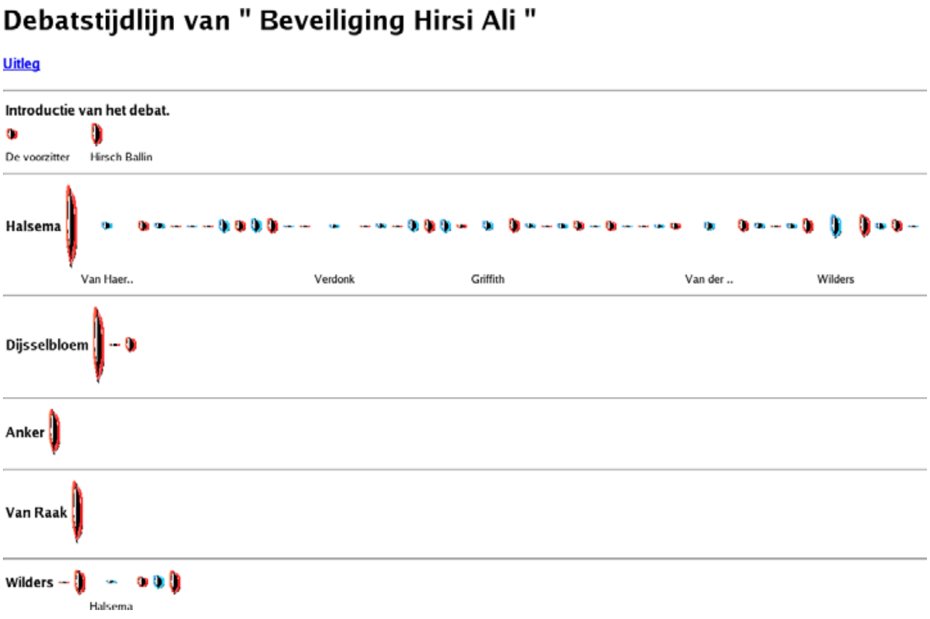


Fig. 4. High-level visualization of the first part of the debate on the protection of Hirsi-Ali. Original available at `http://www.geencommentaar.nl/parlando/index.php?action=doc&filename=HAN8183A16`. The first speaker on the lectern is *Halsema* who is interrupted by *Van Haersma Buma, Verdonk, Griffith, Van der Staai* and *Wilders*, in that order. Only the first time a speaker interrupts, her name is shown.

2. Zooming in on a block, if `A` is the speaker at the lectern and `B,C,D` are the ones interrupting `A`, then blocks very often look like `(AB)+(AC)+(AD)+A`, i.e., a sequence of small conversations with different members with `A` having the last word.

Debates in the Dutch parliament are governed by a set of written regulations and a set of unwritten codes. Both observations above are instantiations of unwritten codes. The first observation restates the rule that the speaker at the lectern always has the last word. The second observation corresponds to the rule that a member of parliament can only have one block of interruptions of a member at the central lectern. See [65] for these rules. Another rule is that someone may only interrupt another 3 times in a row. So according to these unwritten codes the second regular expression should be `(AB){1,3}(AC){1,3}(AD){1,3}A` and none of `B,C,D` should be equal.

Formalizations of these written and unwritten rules in terms of regular expressions, and using these to find *violations* is an interesting open direction of research.[12]

This internal structure of blocks can be used to create high-level overviews of debates which show who attacks who and which can be used for navigation. We present an example in Section 4.3. The regular expression which best fits or describes a block can be obtained by the algorithm which induces DTD's from a set of example XML-files described in [8].

4.3 Exemelification: From Flat PDF to Deep XML

Figure 5 gives a good indication of the mappings created in the exemelification process. The following technique is used. First we extract the text from the PDF using the open source program `pdftohtml`[13] with the `-xml` option. This yields an XML file with for each line of text four coordinates which indicate the bounding box of that text. Multiple columns are detected and preserved. Some font and layout information is preserved but not all. The XML structure is simple and flat:

```
root ⟶ (page)*
page ⟶ (text)*
text ⟶ (#PCDATA,b,i)*
```

On these XML files we use patterns written as regular expressions to add special empty XML elements on places where in the final file an XML element needs to be opened. For instance, the □ is replaced by `<blockstart/>`. A phrase like

Mevrouw **Swenker** (VVD):

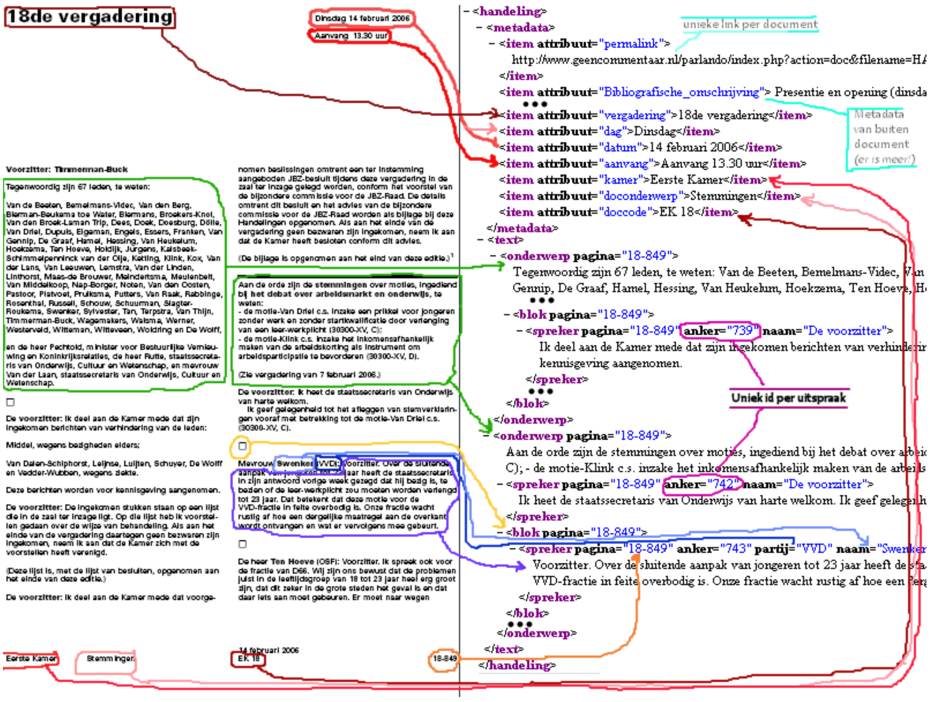
is replaced by

```
<speechstart speaker='Swenker' party='VVD' ... />,
```

with the ... containing additional information.

[12] We have found such violations with Dutch members of parliament who have a new debating style like Wilders and Verdonk.

[13] `http://pdftohtml.sourceforge.net/`



```
- <handeling>
  - <metadata>
    - <item attribuut="permalink">
        http://www.geencommentaar.nl/parlando/index.php?action=doc&filename=HA
      </item>
      <item attribuut="Bibliografische_omschrijving"> Presentie en opening (dinsda
        •••
      <item attribuut="vergadering">18de vergadering</item>
      <item attribuut="dag">Dinsdag</item>
      <item attribuut="datum">14 februari 2006</item>
      <item attribuut="aanvang">Aanvang 13.30 uur</item>
      <item attribuut="kamer">Eerste Kamer</item>
      <item attribuut="doconderwerp">Stemmingen</item>
      <item attribuut="doccode">EK 18</item>
    </metadata>
  - <text>
    - <onderwerp pagina="18-849">
        Tegenwoordig zijn 67 leden, te weten: Van de Beeten, Bemelmans-Videc, Van
        Gennip, De Graaf, Hamel, Hessing, Van Heukelum, Hoekzema, Ten Hoeve, H
          •••
      - <blok pagina="18-849">
        - <spreker pagina="18-849" anker="739" naam="De voorzitter">
            Ik deel aan de Kamer mede dat zijn ingekomen berichten van verhinderir
            kennisgeving aangenomen.
          </spreker>
          •••
        </blok>
      </onderwerp>
    - <onderwerp pagina="18-849">
        Aan de orde zijn de stemmingen over moties, ingediend bij het debat over arbeid
        C); - de motie-Klink c.s. inzake het inkomensafhankelijk maken van de arbeids
      - <spreker pagina="18-849" anker="742" naam="De voorzitter">
          Ik heet de staatssecretaris van Onderwijs van harte welkom. Ik geef gelegenh
        </spreker>
      - <blok pagina="18-849">
        - <spreker pagina="18-849" anker="743" partij="VVD" naam="Swenker
            Voorzitter. Over de sluitende aanpak van jongeren tot 23 jaar heeft de sta
            VVD-fractie in feite overbodig is. Onze fractie wacht rustig af hoe een Berg
          </spreker>
        </blok>
        •••
      </onderwerp>
    </text>
  </handeling>
```

Fig. 5. Example of the mapping from the description of a debate in PDF to the version in XML. Note how the start of a new block is indicated by a □ (mapping indicated in yellow.)

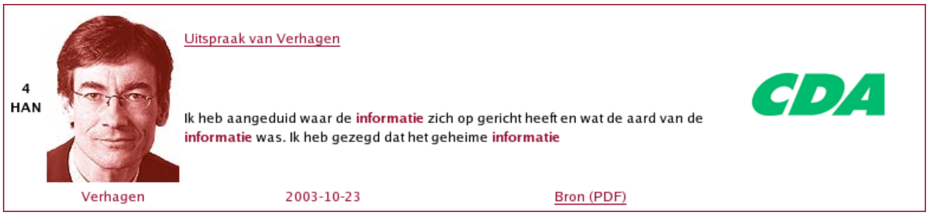
The result of this search and replace process is again a well formed XML file with a similar flat structure as before. In the last step we perform a cascade of groupings starting with the elements which need to be most deeply nested: the paragraphs (the XML elements with tag p). XSLT 2.0 has a very useful command for this task: `xsl:for-each-group`. This command, new in XSLT 2.0, replaces the so-called Muenchian method which was needed in version 1.0 of XSLT [39].

4.4 Applications of the XML Structure

We describe four applications of the XML structure. None of these is possible when working with the PDF data. They are entry point retrieval and the use of permalinks, complex content and structure queries, automatic creation of tables of contents and navigation menus and finally savings on bandwidth.

Entry point retrieval and permalinks. The most natural answer unit in a retrieval system for parliamentary debates is the speech. The result page after a keyword query then will be a ranked list of items consisting of

– the name of the speaker,
– her party,


4
HAN

Uitspraak van Verhagen

Ik heb aangeduid waar de **informatie** zich op gericht heeft en wat de aard van de **informatie** was. Ik heb gezegd dat het geheime **informatie**

Verhagen 2003-10-23 Bron (PDF)

Fig. 6. Answer snippet from result list: photograph of the speaker linking to his bio, logo of his party, a link to the official PDF source, the first 100 characters of his speech and a link to the speech

- a photo of the speaker,
- the date of the speech
- a relevant text snippet of the speech,
- a hyperlink which points to the anchor attached to the speech within a debate, and
- a hyperlink to the original PDF source.

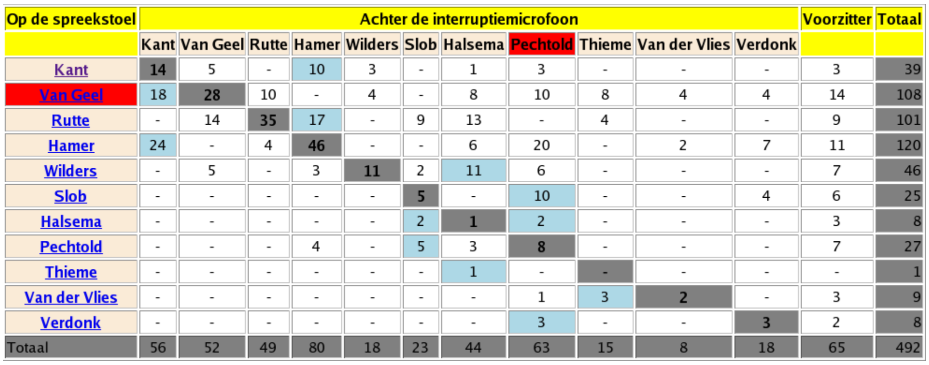
This is how it works in the UK on the site `http://www.theyworkforyou.com`, on the site of the European Parliament, and also in the retrieval engine that we built for the Dutch data `http://www.polidocs.nl`, see Figure 6.

Though natural, this notion of answer is by no means standard for parliamentary retrieval systems. The search systems of the German and Flemish parliaments return the proceedings of one day. These can be PDF files with two columns of up to a 100 pages. In the Netherlands, the situation is even more complex:

- proceedings before 1995 are available at `http://www.statengeneraaldigitaal.nl/`. The answer unit is the proceedings of a complete meeting;
- proceedings after 1995 are available at `http://parlando.sdu.nl/cgi/login/anonymous`. The answer unit roughly corresponds to one topic. It is indeed roughly as topics almost never start at the top of a page nor finish at the bottom of a page, and the PDF documents at Parlando are divided into overlapping sets of pages;
- preliminary proceedings are available at `http://www.tweedekamer.nl/`. Search is not really possible on this site. Preliminary proceedings are available in HTML which is shown together with a navigation menu which contains the same topic–block–speech hierarchy as described in Section 2.

During the transformation from PDF to XML we add a unique anchor ID to every speech. This anchor together with the number of the document given by the parliament constitutes a unique permanent reference to each speech.

The permanent hyperlinks (permalinks) for each speech made in parliament have many applications besides making entry point retrieval possible. Examples are easy referencing in emails, weblogs and even scientific papers. Permalinks also stimulate third party development of websites (like mashups) based on this data.



Op de spreekstoel	Achter de interruptiemicrofoon											Voorzitter	Totaal
	Kant	Van Geel	Rutte	Hamer	Wilders	Slob	Halsema	Pechtold	Thieme	Van der Vlies	Verdonk		
Kant	14	5	-	10	3	-	1	3	-	-	-	3	39
Van Geel	18	28	10	-	4	-	8	10	8	4	4	14	108
Rutte	-	14	35	17	-	9	13	-	4	-	-	9	101
Hamer	24	-	4	46	-	-	6	20	-	2	7	11	120
Wilders	-	5	-	3	11	2	11	6	-	-	-	7	46
Slob	-	-	-	-	-	5	-	10	-	-	4	6	25
Halsema	-	-	-	-	-	2	1	2	-	-	-	3	8
Pechtold	-	-	-	4	-	5	3	8	-	-	-	7	27
Thieme	-	-	-	-	-	-	1	-	-	-	-	-	1
Van der Vlies	-	-	-	-	-	-	-	1	3	2	-	3	9
Verdonk	-	-	-	-	-	-	-	3	-	-	3	2	8
Totaal	56	52	49	80	18	23	44	63	15	8	18	65	492

Fig. 7. Who attacks who in the debate *Algemene Beschouwingen* on September 17 2008. Speakers at the lectern are listed in the first column; their attackers on the top row. The numbers in the cell indicate how often the person on the x-axis interrupted the speech by the person on the y-axis. The numbers on the diagonal (in gray) are the number of answers to interruptions given by the speaker on the lectern. Source: `http://staff.science.uva.nl/~marx/politicalmashup/AB2008/DebatstructuurAB2008.html`.

Complex content and structure queries. The explicit XML structure allows one to formulate information needs using natural XPath, XQuery, XSLT or NEXI [37,51] expressions. We illustrate this by some examples:

- *give speeches about Islam from debates about immigration* can be formulated as the NEXI query
  `//topic[about(.,immigration')]//speech[about(.,'islam')].`
- *give all speakers who interrupted Geert Wilders during the Islam debate* can be formulated in XPath 1.0 as
  `//topic[@title='islam']//block[@speaker='Wilders']//speech[@speaker != 'Wilders']/@speaker.`
- *give a list of these speakers together with their number of interruptions ordered by that number* is expressed in XQuery or XSLT using the structure of the XPath expression from the last bullet and the `fn:count()` function.
- *Create a cross table of speakers at the lectern and their interrupters and list the number of interruptions in each data cell* is a typical task for XSLT. The result for the *Algemene Beschouwingen* on September 17 2008, containing 624 speeches in one debate, is reproduced in Figure 7.

Based on experience with bachelor information science students we claim that it is easier to formulate such complex queries in XSLT directly on the original XML files than to state them in SQL on a relational representation of a debate.

Automatic creation of tables of contents and navigation menus. The notes of a one day meeting of Parliament tend to be quite long, typically between 50 and 100 pages two column PDF. Within the current search engine at `www.statengeneraaldigitaal.nl` these are the documents returned to users. Unfortunately these documents do not contain a table of contents listing the topics discussed in a meeting. But even if such tables would be available in PDF they would be of little help when browsing these documents on a computer because they do not contain hyperlinks.

Since the topics are explicit elements in the XML version of the data it is straightforward to automatically generate a hyperlinked table of contents for each document. This can be done with XSLT.

Even one topic can be quite long. For instance, the meeting of September 18, 2008 took the whole day, consisted of 624 speeches with a total of 74068 words, all within one topic. Fortunately the block structure can be used to break up this large chunk of text. In fact the debate timelines in Figure 4 are navigation menus: each mouth contains a hyperlink to exactly that part of the proceedings which record the speech represented by the mouth. Again this is possible due to the added anchors.

Savings on bandwidth. The Dutch parliamentary data from before 1995 was only available in printed form. Within the StatenGeneraalDigitaal project of the Dutch Royal Library this data is scanned and OCR-ed (Optical Character Recognition), resulting in complex PDF documents consisting of facsimile images of every page, the OCR-ed text and a mapping from each word to its position on every page[14].

Such files can be enormous in size. For instance, the proceedings on `http://resolver.kb.nl/resolve?urn=sgd:mpeg21:19851986:0000761` are 72 pages PDF. The size of this file is 24 Megabyte. The same proceedings in XML is less than .5Mb. We experimented with reducing the size with gzip: the PDF became 23Mb and the XML was reduced to 156Kb. This is 0.65% of the size of the original PDF.

Preliminary experiments show that using XSLT and LaTeX the original format of the proceedings can be produced with very good layout accuracy and very fast. The resulting PDF is again less than .5Mb. Producing this PDF from the gzipped XML can even be done at query time: on a standard Linux box this process took less than 1.5 seconds real time. For detailed information on this experiment see `http://ilps.science.uva.nl/PoliticalMashup/2008/10/trading-space-for-time`.

Thus large savings in bandwidth and storage space become possible. We must note that the XML version is based on OCR-ed data and contains quite a few OCR errors. Of course these come back in the PDF created from the XML source. Repairing such mistakes automatically has been done with promising accuracy by Martin Reyneart using his TICL technique [53].

We believe that the facsimiles need to be available as the ultimate source but that in a search and browse interaction process with the data the alternative, much smaller, version based on the XML is preferable. Users get results faster, they get clean hyperlinked files, and they use much less bandwidth. Once a user knows exactly which document she wants to consult, the large facsimile PDF can be downloaded.

4.5 To Conclude

We have shown that text extraction from Parliamentary proceedings based on regular expressions and XSLT is feasible, scalable, possible on both digital and scanned data, and leads to numerous benefits.

[14] See `http://www.statengeneraaldigitaal.nl/backgrounds.html` for extensive information on the digitization process (in Dutch).

We stress that this extraction process is transparent, repeatable and independent of any software or hardware because we only use declarative programming languages with a well described semantics. This means that when the extraction scripts (which are themselves XML files, since it is XSLT) together with a copy of the XSLT reference [39] are stored together with the original digitized data in a safe place, it is in principle always possible to recreate the XML versions we have described here.

Several parliaments are digitizing their complete historical data. We are aware of efforts in the UK, Ireland, Australia, and the Flemish Parliament. Our DTD is general enough to fit all these proceedings. This opens the possibility of creating a huge integrated multi-lingual XML repository of parliamentary proceedings. Such a repository will facilitate comparative parliamentary (historical) research.

5 Operationalization of Policy Framing Questions on Parliamentary Data with XQuery

This section is based on joint work with Loredana Afanasiev [3].

Even though it is often much easier to express a user's information need in XPath and XQuery than in SQL, the whole process may still be quite complex. Here we present an elaborate example which involved a lot of data preprocessing. Having done that, writing the actual query was rather simple.

The Information Need

An important field of study in political science is concerned with agenda-setting: Who puts an issue on the political agenda, using which media? And then, how does an issue evolve over time? Who takes the lead: do the media follow debate in parliament or is it the other way around? This question is being researched in [54] for a period of around 20 years for a number of hot issues like *"islamic threat and terrorism"*. The authors of [54] compiled their data "by hand" from PDF sources using Google style keyword queries. The proceedings of the Dutch parliament are now available in XML [25], so we could restate the information need in XPath and XQuery.

Here we copy how an information need can be operationalized in XQuery and visualized using Google charts. Figure 8 contains part of one of the XML documents in the collection that was asked. The data is in Dutch but we translate the most important parts. An elaborate description of the operationalization process in given on `http://ilps.science.uva.nl/PoliticalMashup/framing-questions-on-polidocs-data`. Here we summarize part of it.

The researcher gave us the following description of the query for his "anti-islam" frame:

```
(islam* AND (bedreiging* OR terrorisme)).
```

He wanted to have for each year, all speeches made in parliament which matched his query. From those speeches he wanted to know the date, the speaker, his/her party, and of course the text of the speech. In addition we provided a URL which links exactly to this speech. Here is how the output then looked like:

```
<frame name="anti-islam">
 <collection name="HAN">
  <result>
    <date>14-03-1995</date>
    <politicus party="VVD" name="Talsma"/>
    <polidocslink>
      http://www.polidocs.nl/XML/HAN/HAN2346.xml#665
    </polidocslink>
    <content>
Mevrouw de voorzitter! Ik zal over drie punten
opmerkingen maken, er daarbij
...
    </content>
  </result>
...
 </collection>
</frame>
```

The XQuery to create this output is given in Figure 9. It is now easy to write XQueries generating useful statistics, like the number of hits per year. One can even generate the code for the plot using XQuery. Here this is done in two steps: the first query contains the logical expression and stores it in an XML format. The second query reformats the output into the specific format required by Google charts. The queries are in Figure 10 and 11. The intermediate XML looks as follows

```
<res>
  <frame name="anti-islam">
     <year name="1986">1</year>
     <year name="1988">1</year>
     <year name="1989">4</year>

      ...
</res>
```

The final plot and the code to produce it is in Figure 12.

Acknowledgements. Thanks are due to my coauthors of the presented material: Loredana Afanasiev, Balder ten Cate and Tim Gielissen [3,4,25,64].

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="showHAN.xsl" type="text/xsl"?>


<handeling>
<metadata>


.....

<item attribuut="Vindplaats">
Handelingen 2006-2007, nr. 93, Tweede Kamer, pag. 5260-5319
</item>
<item attribuut="Afkomstig_van">Staten Generaal (SG)</item>
<item attribuut="Datum_vergadering">06-09-2007</item>
<item attribuut="Document-id">HAN8168A06</item>

....

</metadata>

<text>

<onderwerp pagina="93-5260">

Aan de orde is het debat over het kabinetsstandpunt
ten aanzien van het rapport Dynamiek in
islamitisch activisme van de Wetenschappelijke
Raad voor het Regeringsbeleid (WRR) (30800-VI,
nr. 115).

<blok pagina="93-5260">


<spreker pagina="93-5260" anker="2" partij="PVV" naam="Wilders">
 Mevrouw de voorzitter. Om te
beginnen mijn oprechte dank aan u persoonlijk omdat u
op mijn verjaardag vandaag een debat over de islam
heeft gepland. Een mooier cadeau had ik mij niet kunnen

.....
```

Fig. 8. Part of the Dutch parliamentary proceedings in XML

```
  (: Frame Anti-Islam
    (islam* AND (bedreiging* or terrorisme))

   Model: expressed via the contains() function of XQuery
:)

<frame name="anti-islam">
{

(: Collection HAN :)
<collection name="HAN">
{
for $d in collection('HAN')
  for $spreker in $d//spreker[contains(string(.), "islam") and
                              ( contains(string(.), "bedreiging") or
                                contains(string(.), "terrorisme")) ]
  return
    <result>
    {
     <date>{$d//metadata/item[@attribuut="Datum_vergadering"]/text()}
     </date>,
     <politicus name="{data($spreker/@naam)}"
                party="{data($spreker/@partij)}"/>,
     <polidocslink>{
       concat(
        "http://www.polidocs.nl/XML/HAN/",
        $d//metadata/item[@attribuut="Document-id"]/text(),
        ".xml#",
        $spreker/@anker
         )
     }</polidocslink>,
     <content>{string($spreker)}</content>
    }
    </result>
}
</collection>

}
</frame>
```

Fig. 9. XQuery to generate the anti-islam frame

```
(:
 Per frame per year return the count hits
:)

declare function local:get-year($d as element()) as xs:string* {
  substring-before(string($d),'-')
      [string-length(.)=4 and not(contains(.,'-'))],
  substring-before(string($d),'.')
      [string-length(.)=4 and not(contains(.,'.'))],
  substring-after(string($d),'-')
      [string-length(.)=4 and not(contains(.,'-'))],
  substring-after(string($d),'.')
      [string-length(.)=4 and not(contains(.,'.'))]
};

<res>
{
for $f in doc('Frames.xml')//frame
let $years :=
  for $d in $f//date return local:get-year($d)
return
<frame>
{
  $f/@name,
  for $y in distinct-values($years)
  order by $y
  return
  <year name="{$y}">{
    count($years[. eq $y])
  }
  </year>
}
</frame>
}
</res>
```

Fig. 10. XQueries to produce the yearly aggregates

```
(:
 Per frame per year return the count hits
:)

let $frames := doc('agg2.xml')//frame
let $allyears :=
  for $y in distinct-values($frames/year/@name) order by $y return $y
let $minyear := min($allyears)
let $maxyear := max($allyears)
let $maxhit := max($frames/year)
let $data := string-join(
    for $f in $frames
    return concat(string-join($allyears,','),'|',string-join(
      for $y in $allyears
      return if($f/year[@name = $y])
             then ($f/year[@name = $y])
             else('0'),',')
                                                            ),
          '|')
let $ds := string-join(
  for $f in $frames
  return
    concat(string($minyear),',',string($maxyear),',','0,',
           string($maxhit)),",")

return
concat(
'http://chart.apis.google.com/chart?',
'chs=900x200&amp;',
'cht=lxy&amp;',
'chxt=x,y&amp;',
'chtt=Frame+hits+per+year+(HAN,+KVR,+MOT)&amp;',
'chds=',$ds,'&amp;',
'chd=t:',$data,'&amp;',
'chxr=0,',string($minyear),',',string($maxyear),',1|1,0,',
          string($maxhit),',5&amp;',
'chdl=',string-join($frames/@name,'|'),'&amp;',
'chco=FF0000,00FF00,0000FF,000000&amp;',
'chm=s,FF0000,0,-1,5|s,00FF00,1,-1,5|s,0000FF,2,-1,5|s,000000,3,-1,5&amp;'
)
```

Fig. 11. XQuery to produce the Google Chart plot

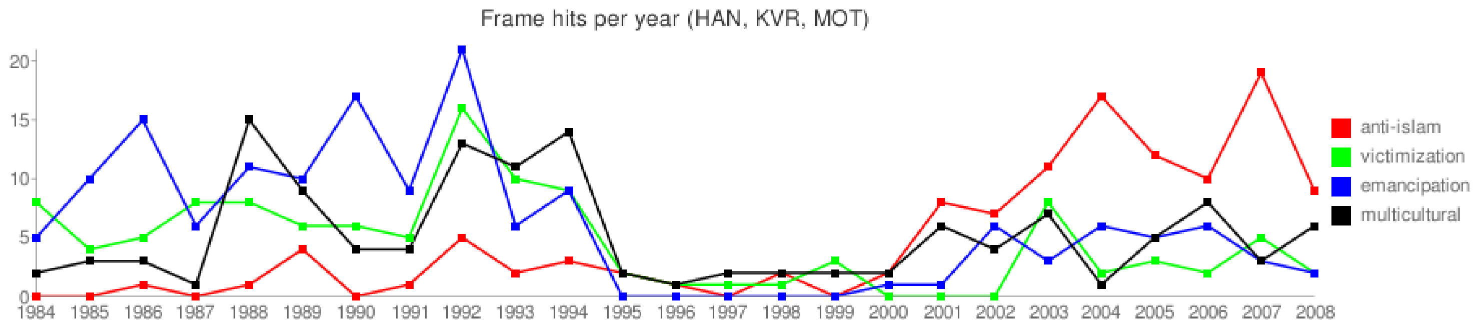


Fig. 12. Frame hits per year, for for frames. Source: http://chart.apis.google.com/chart?chs=900x200&cht=lxy&chxt=x,y&chtt=Frame+hits+per+year+(HAN,+KVR,+MOT)&chds=1984,2008,0,21,1984,2008,0,21,1984,2008,0,21,1984,2008,0,21&chd=t:1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008|0,0,1,0,1,4,0,1,5,2,3,2,1,0,2,0,2,8,7,11,17,12,10,19,9|1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008|8,4,5,8,8,6,6,5,16,10,9,2,1,1,1,3,0,0,0,8,2,3,2,5,2|1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008|5,10,15,6,11,10,17,9,21,6,9,0,0,0,0,0,1,1,6,3,6,5,6,3,2|1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008|2,3,3,1,15,9,4,4,13,11,14,2,1,2,2,2,2,6,4,7,1,5,8,3,6&chxr=0,1984,2008,1|1,0,21,5&chdl=anti-islam|victimization|emancipation|multicultural&chco=FF0000,00FF00,0000FF,000000&chm=s,FF0000,0,-1,5|s,00FF00,1,-1,5|s,0000FF,2,-1,5|s,000000,3,-1,5&

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the web. Morgan Kaufman, San Francisco (2000)
2. Afanasiev, L., Franceschet, M., Marx, M., Zimuel, E.: XCheck: a Platform for Benchmarking XQuery Engines. In: Proceedings of VLDB, Demo, Seoul, Korea. ACM Press, New York (2006)
3. Afanasiev, L., Marx, M.: Operationalization of policy framing questions on parliamentary data with XQuery (2009), `http://ilps.science.uva.nl/PoliticalMashup/framing-questions-on-polidocs-data/`
4. Afanasiev, L., ten Cate, B., Marx, M.: Lekker bomen. Nieuwsbrief van de NVTI 11, 38–52 (2007)
5. Axyana software. Qizx/open. An open-source Java implementation of XQuery (2006), `http://www.axyana.com/qizxopen`
6. Balog, K.: People Search in the Enterprise. PhD thesis, University of Amsterdam (June 2008)
7. Benedikt, M., Fan, W., Kuper, G.M.: Structural properties of XPath fragments. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 79–95. Springer, Heidelberg (2002)
8. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from xml data. In: WWW 2008: Proceeding of the 17th international conference on World Wide Web, pp. 825–834. ACM, New York (2008)
9. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)
10. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: PODS, pp. 10–19 (2006)
11. Brüggemann-Klein, A., Wood, D.: Caterpillars, context, tree automata and tree pattern matching. In: Rozenberg, G., Thomas, W. (eds.) Proceedings of DLT 1999: Foundations, Applications and Perspectives, pp. 270–285. World Scientific Publishing, Singapore (2000)
12. Chandra, A., Harel, D.: Structure and complexity of relational queries. J. Comput. Syst. Sci. 25(1), 99–128 (1982)
13. Clark, J., DeRose, S.: XML Path Language (XPath), `http://www.w3.org/TR/xpath`
14. Clarke, E.M., Schlingloff, B.-H.: Model checking. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1367–1522. Elsevier Science Publishers, Amsterdam (2000)
15. Cleaveland, R., Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus. Form. Methods Syst. Des. 2(2), 121–147 (1993)
16. Codd, E.: Relational completeness of data base sublanguages. In: Rustin, R. (ed.) Database Systems, pp. 33–64. Prentice-Hall, Englewood Cliffs (1972)
17. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory. Springer, Heidelberg (1995)
18. Engelfriet, J., Hoogeboom, H.J.: Nested pebbles and transitive closure. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 477–488. Springer, Heidelberg (2006)
19. Etessami, K., Vardi, M.: First-order logic with two variables and unary temporal logic. In: Proc. LICS 1997, pp. 228–235 (1997)
20. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: SMOQE: a system for providing secure access to XML. In: Proceedings VLDB 2006, pp. 1227–1230 (2006)
21. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Rewriting regular XPath queries on XML views. In: Proceedings ICDE 2007 (2007)
22. Fernández, M., Siméon, J., Chen, C., Choi, B., Gapeyev, V., Marian, A., Michiels, P., Onose, N., Petkanics, D., Ré, C., Stark, M., Sur, G., Vyas, A., Wadler, P.: Galax. The XQuery implementation (2006), `http://www.galaxquery.org`

23. Filiot, E., Niehren, J., Talbot, J.-M., Tison, S.: Polynomial time fragments of xpath with variables. In: Proceedings of PODS 2007 (2007)
24. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18, 194–211 (1979)
25. Gielissen, T., Marx, M.: Exemelification of parliamentary debates. In: Proceedings of the 9th Dutch-Belgian Information Retrieval Workshop (DIR 2009), Twente, The Netherlands, pp. 19–25 (2009)
26. Goris, E., Marx, M.: Looping caterpillars. In: Proceedings LICS 2005. IEEE Computer Society, Los Alamitos (2005)
27. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. In: VLDB 2002 (2002)
28. Gottlob, G., Koch, C., Schulz, K.: Conjunctive queries over trees. In: Proceedings PODS 2004, pp. 189–200 (2004)
29. Grohe, M., Schweikardt, N.: The succinctness of first-order logic on linear orders 1(1) (2005)
30. Hajer, M.: Setting the stage, a dramaturgy of policy deliberation. Administration & Society 36(6), 624–647 (2005)
31. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: The teenage years. In: Dayal, U., Whang, K.-Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.-K. (eds.) VLDB, pp. 9–16. ACM, New York (2006)
32. Halpern, J.Y., Harper, R., Immerman, N., Kolaitis, P.G., Vardi, M.Y., Vianu, V.: On the unusual effectiveness of logic in computer science. The Bulletin of Symbolic Logic 7(2), 213–236 (2001)
33. Hariman, R.: Political style. The artistry of power. University of Chicago Press, Chicago (1995)
34. Hodkinson, I., Reynolds, M.: Separation - past, present, and future. In: Artemov, S., et al. (eds.) We will show them! (Essays in honour of Dov Gabbay on his 60th birthday), pp. 117–142. College Publications (2005)
35. Hollenberg, M.: An equational axiomatization of dynamic negation and relational composition. Journal of Logic, Language and Information 6(4), 381–401 (1997)
36. Kamp, J.A.W.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles (1968)
37. Kamps, J., Marx, M., de Rijke, M., Sigurbjörnsson, B.: Articulating information needs in XML query languages. ACM Trans. Inf. Syst. 24(4), 407–436 (2006)
38. Kay, M.: XPath 2.0 Programmer's Reference. Wrox (2004)
39. Kay, M.: XSLT 2.0 3rd edn. Programmer's Reference. Wrox (2004)
40. Kay, M.H.: SaxonB. An XSLT and XQuery processor (2006), `http://saxon.sourceforge.net`
41. Lange, M.: Model checking propositional dynamic logic with all extras. Journal of Applied Logic 4(1), 39–49 (2005)
42. Lazer, D., Pentland, A., Adamic, L., Aral, S., Barabasi, A.-L., Brewer, D., Christakis, N., Contractor, N., Fowler, J., Gutmann, M., Jebara, T., King, G., Macy, M., Roy, D., Van Alstyne, M.: Computational social science. Science 323(5915), 721–723 (2009)
43. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. PODS, pp. 233–246 (2002)
44. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogeneous information sources using source descriptions. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) VLDB, pp. 251–262. Morgan Kaufmann, San Francisco (1996)
45. Lutz, C.: The Complexity of Reasoning with Concrete Domains. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen (2002)
46. Marx, M.: XPath with conditional axis relations. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 477–494. Springer, Heidelberg (2004)

47. Marx., M.: Conditional XPath. ACM Transactions on Database Systems 30(4), 929–959 (2005)
48. Marx, M., de Rijke, M.: Semantic Characterizations of Navigational XPath. ACM SIGMOD Record 34(2), 41–46 (2005)
49. MonetDB/XQuery. An XQuery Implementation (2006), `http://monetdb.cwi.nl/XQuery`
50. Neven, F., Schwentick, T.: XPath containment in the presence of disjunction, DTDs, and variables. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 312–326. Springer, Heidelberg (2002)
51. O'Keefe, R.A., Trotman, A.: The Simplest Query Language That Could Possibly Work. In: Proceedings of the 2nd INEX Workshop (2004)
52. Rahm, E., Do, H.-H.: Data cleaning: Problems and current approaches. IEEE Techn. Bulletin on Data Engineering 23(4) (2000)
53. Reynaert, M.: Non-interactive OCR post-correction for giga-scale digitization projects. In: Gelbukh, A. (ed.) CICLing 2008. LNCS, vol. 4919, pp. 617–630. Springer, Heidelberg (2008)
54. Roggeband, C., Vliegenthart, R.: Divergent framing: The public debate on migration in the Dutch parliament and media, 1995-2004. West European Politics 30(3), 524–548 (2007)
55. Runapongsa, K., Patel, J.M., Jagadish, H.V., Al-Khalifa, S.: The michigan benchmark: A microbenchmark for XML query processing systems. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 160–161. Springer, Heidelberg (2003)
56. Seaton, J.: The Scottish Parliament and e-democracy. Aslib Proceedings: New Information Perspectives 57(4), 333–337 (2005)
57. Sigurbjörnsson, B.: Focused information access using XML element retrieval. PhD thesis, University of Amsterdam (2006)
58. Stockmeyer, L.: The Complexity of Decision Problems in Automata Theory. PhD thesis, Dept. Electrical Engineering. MIT, Cambridge (1974)
59. Tarski, A.: On the calculus of relations. Journal of Symbolic Logic 6, 73–89 (1941)
60. Tarski, A., Givant, S.: A Formalization of Set Theory without Variables, vol. 41. AMS Colloquium publications, Providence (1987)
61. ten Cate, B.: The expressivity of XPath with transitive closure. In: Proceedings of PODS 2006, pp. 328–337 (2006)
62. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of XPath 2.0. In: Proceedings PODS 2007 (2007)
63. ten Cate, B.D., Marx, M.: Axiomatizing the logical core of xPath 2.0. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 134–148. Springer, Heidelberg (2006)
64. ten Cate, B., Marx, M.: Navigational xpath: calculus and algebra. SIGMOD Record 36(2), 19–26 (2007)
65. van Baalen, C., Bos, A.: In vergadering bijeen. Rituelen, symbolen, tradties en gebruiken in de Tweede Kamer. In: Jaarboek Parlementaire Geschiedenis 2008, Boom (2008)
66. van Benthem, J.: Program constructions that are safe for bisimulation. Studia Logica 60(2), 330–331 (1998)
67. Vardi, M.: On the complexity of bounded–variable queries. In: Proceedings PODS 1995, pp. 266–276 (1995)
68. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: Proceedings of STOC 1982, pp. 137–146. ACM Press, New York (1982)

Appendix: Until Queries

The following equivalent queries were used in the experiment reported in Figure 3. The `id` attribute of the `query` element gives the query name used in the histogram; the description element gives a short query description; and the syntax element contains the actual query in XQuery.

```
<queries>

<query id="U1">
<description>Use recursive function</description>
<syntax engine="all"><![CDATA[

  declare namespace my='my-functions';
  declare function my:until($input as node()*)
  {
    for $i in $input return
    ($i/child::*[@aFour=3],my:until($i/child::*[@aFour=0]))
  };
  let $start := doc("mbench.xml")//*[@aLevel=5]
  return
  my:until($start)

]]></syntax>
</query>
<query id="U2">
<description>Use except</description>
<syntax engine="all"><![CDATA[

  let $start := doc("mbench.xml")//*[@aLevel=5]
  return
  $start//*[@aFour=3] except $start//*[not(.[@aFour=0])]//*

]]></syntax>
</query>
<query id="U3">
<description>Forward every with if </description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
  if (every $inbetween in  $start//*[.//*[. is $end]] satisfies
        $inbetween/@aFour=0)
  then $end
  else ()

]]></syntax>
</query>
<query id="U4">
<description>Forward as in the usual FO writing of until
             but now with a predicate
</description>
```

```
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
    $end[every $inbetween in $start//*[.//*[. is $end]] satisfies
    $inbetween/@aFour=0]

]]></syntax>
</query>
<query id="U5">
<description>Forward empty with if </description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5],
      $end in $start//*[@aFour=3]
  return
  if (empty($start//*[not(.[@aFour=0])]//*[. is $end]))
  then $end
  else ()

]]></syntax>
</query>
<query id="U6">
<description>Backward with every</description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5]
  return
  $start//*[@aFour=3]
    [every $inbetween in ancestor::*[ancestor::*[. is $start]]
      satisfies $inbetween/@aFour=0]

]]></syntax>
</query>
<query id="U7">
<description>Backward with predicate</description>
<syntax engine="all"><![CDATA[

  for $start in doc("mbench.xml")//*[@aLevel=5]
  return
  $start//*[@aFour=3]
    [not(ancestor::*[not(.[@aFour=0])]/ancestor::*[. is $start])]

]]></syntax>
</query>

</queries>
```

Foundations of RDF Databases

Marcelo Arenas[1], Claudio Gutierrez[2], and Jorge Pérez[1]

[1] Department of Computer Science, Pontificia Universidad Católica de Chile
[2] Department of Computer Science, Universidad de Chile

Abstract. The goal of this paper is to give an overview of the basics of the theory of RDF databases. We provide a formal definition of RDF that includes the features that distinguish this model from other graph data models. We then move into the fundamental issue of querying RDF data. We start by considering the RDF query language SPARQL, which is a W3C Recommendation since January 2008. We provide an algebraic syntax and a compositional semantics for this language, study the complexity of the evaluation problem for different fragments of SPARQL, and consider the problem of optimizing the evaluation of SPARQL queries, showing that a natural fragment of this language has some good properties in this respect. We furthermore study the expressive power of SPARQL, by comparing it with some well-known query languages such as relational algebra. We conclude by considering the issue of querying RDF data in the presence of RDFS vocabulary. In particular, we present a recently proposed extension of SPARQL with navigational capabilities.

1 Introduction

The Resource Description Framework (RDF) [34] is a data model for representing information about World Wide Web resources. Jointly with its release in 1998 as Recommendation of the W3C, the natural problem of querying RDF data was raised. Since then, several designs and implementations of RDF query languages have been proposed. In 2004, the RDF Data Access Working Group, part of the W3C Semantic Web Activity, released a first public working draft of a query language for RDF, called SPARQL [45]. Since then, SPARQL has been rapidly adopted as the standard for querying Semantic Web data. In January 2008, SPARQL became a W3C Recommendation.

RDF and SPARQL are two of the core technologies in the data and query layers of the *Semantic Web stack*. In this paper, we give an overview of the current state of the theory of RDF and SPARQL from a database perspective. We first provide a formal definition of RDF that includes the features that distinguish this model from other database models. We then move into the fundamental issue of querying RDF data with SPARQL. We provide an algebraic syntax and a compositional semantics for this language, study the complexity of the evaluation problem for different fragments of SPARQL, and consider the problem of optimizing the evaluation of SPARQL queries, showing that a natural fragment of this language has some good properties in this respect. We furthermore study the expressive power of SPARQL, by comparing it with some well-known query languages such as relational algebra. We conclude by considering the issue of querying RDF data in the presence of RDFS vocabulary. In particular, we present a recently proposed extension of SPARQL with navigational capabilities, and show that this language is expressive enough to deal with the semantics of the RDFS vocabulary.

The paper is organized as follows. In Section 2, we introduce RDF as a data model. In Section 3, we provide a formalization of the syntax and semantics of SPARQL. In Section 4, we study the complexity of the evaluation problem for SPARQL and some optimization results for this language. In Section 5, we study the expressiveness of SPARQL. Finally, we present in Section 6 an extension of SPARQL that gives navigational capabilities to the language and allows to deal with the RDFS vocabulary.

2 The RDF Data Model

The Semantic Web is a proposal to build an infrastructure of machine-readable semantics for the data on the Web. In 1998, the W3C issued a recommendation of a metadata model and language to serve as the basis for such infrastructure, the *Resource Description Framework (RDF)* [32]. As RDF evolves, it is increasingly gaining attraction from both researchers and practitioners, and is being implemented in world-wide initiatives such as the Open Directory Project [39], Dublin Core [48], FOAF [49], and RSS [46].

RDF follows the W3C design principles of interoperability, extensibility, evolution and decentralization. Particularly, the RDF model was designed to have a simple data model, with a formal semantics and provable inference, with an extensible URI-based vocabulary, and which allows anyone to make statements about any resource. In the RDF model, the universe to be modeled is a set of *resources*, essentially anything that can have a *universal resource identifier*, URI [50]. The language to describe them is a set of *properties*, technically binary predicates. Descriptions are *statements* very much in the subject-predicate-object structure, where predicate and object are resources or strings. Both subject and object can be anonymous objects, known as *blank nodes.* In addition, the RDF specification includes a built-in vocabulary with a normative semantics (RDFS). This vocabulary deals with inheritance of classes and properties, as well as typing, among other features [11].

The RDF model is specified in a series of W3C documents [11,27,32,34]. In this section, we introduce an abstract version of the RDF data model, which is both a fragment following faithfully the original specification, and also an abstract version suitable to do formal analysis. What is left out are features of RDF dealing with some implementation issues, such as detailed typing issues, some distinguish vocabulary which has no particular semantics, and all topics involved with the XML-based syntax and serialization. The original formulation of this fragment was introduced in [23], and enriched and corrected in [37]. The main goal of isolating such a fragment is to have a simple and stable core over which to discuss theoretical issues, dealing with RDF from a database point of view.

2.1 RDF Graphs

Assume there are pairwise disjoint infinite sets U (RDF URI references) and B (Blank nodes)[1]. Through the paper we assume U and B fixed, and for simplicity we denote unions of these sets simply concatenating their names. A tuple $(s, p, o) \in UB \times U \times UB$ is called an *RDF triple*. In this tuple, s is the *subject*, p the *predicate*, and o the *object*.

Definition 1. *An* RDF graph *(or simply a graph) is a set of RDF triples. A graph is* ground *if it has no blank nodes.*

Graphically, we represent RDF graphs as follows: each triple (s, p, o) is represented by a labeled edge $s \xrightarrow{p} o$. Notice that the set of arc labels can have a non-empty intersection with the set of node labels. Thus, technically speaking, and "RDF graph" is not a graph in the classical sense (for further discussion on this issue see [26]).

In what follows, we need the fundamental notion of homomorphism. Given two RDF graphs G_1 and G_2, a *homomorphism* $h : G_1 \to G_2$ is a mapping from UB to UB such that $h(u) = u$ for every element $u \in U$, and for every triple (s, p, o) in G_1, it holds that $(h(s), h(p), h(o)) \in G_2$. We denote by $h(G_1)$ the RDF graph $\{(h(s), h(p), h(o)) \mid (s, p, o) \in G_1\}$. Thus, a homomorphism h from G_1 to G_2 is such that $h(G_1) \subseteq G_2$.

2.2 RDFS

The RDF specification includes a set of reserved words, the RDFS vocabulary (RDF Schema [11]), which is designed to describe relationships between resources and properties like attributes of resources (traditional attribute-value pairs). Roughly speaking, this vocabulary can be conceptually divided into the following groups:

(a) A set of *properties*, which are binary relations between subject resources and object resources: rdfs:subPropertyOf (denoted by `sp` in this paper), rdfs:subClassOf (`sc`), rdfs:domain (`dom`), rdfs:range (`range`) and rdf:type (`type`).

(b) A set of classes, that denote set of resources. Elements of a class are known as *instances* of that class. To state that a resource is an instance of a class, the reserved word `type` may be used.

[1] For the sake of simplicity, here we do not make a special distinction between URIs and Literals, and we assume that RDF graphs are constructed by using only URIs and Blank nodes. The inclusion of literals does not change any of the results of this paper.

(c) Other functionalities, like a system of classes and properties to describe lists, and a system for doing reification.
(d) Utility vocabulary used to document, comment, etc. (the complete vocabulary can be found in [11]).

The groups in (b), (c) and (d) have a light semantics, essentially describing their internal relationships in the ontological design of the system of classes of RDFS. Their semantics is defined by a set of "axiomatic triples" [27], which express the relationships among these reserved words. All axiomatic triples are "structural", in the sense that do not refer to external data. Much of this semantics corresponds to what in standard languages is captured via typing.

On the contrary, the group (a) is formed by predicates whose intended meaning is non-trivial, and is designed to relate individual pieces of data external to the vocabulary of the language. Their semantics is defined by rules which involve variables (to be instantiated by actual data). For example, rdfs:subClassOf (`sc`) is a reflexive and transitive binary property; and when combined with rdf:type (`type`) specify that the type of an individual (a class) can be lifted to that of a superclass.

The group (a) forms the core of the RDF language and, from a theoretical point of view, it has been shown to be a very stable core to work with (the detailed arguments supporting this claim are given in [37]). Thus, throughout the paper we focused on the fragment of RDFS given by the set of keywords $\{\texttt{sp}, \texttt{sc}, \texttt{type}, \texttt{dom}, \texttt{range}\}$.

2.3 Semantics of RDF Graphs

In this section, we present the formalization of the semantics of RDF given in [27,37]. The normative semantics for RDF graphs given in [27] follows a standard logical treatment, including classical notions such as model, interpretation, entailment, and so on. We present the simplification of the normative semantics proposed in [37]. It is important to notice that these two approaches were shown to be equivalent for the fragment of the RDFS vocabulary considered in this paper [37].

An RDF interpretation is a tuple $\mathcal{I} = (\mathit{Res}, \mathit{Prop}, \mathit{Class}, \mathit{PExt}, \mathit{CExt}, \mathit{Int})$, where (1) Res is a nonempty set of *resources*, called the *domain* or *universe* of $\mathcal{I}$; (2) Prop is a set of property names (not necessarily disjoint from Res); (3) $\mathit{Class} \subseteq \mathit{Res}$ is a distinguished subset of Res identifying if a resource denotes a class of resources; (4) $\mathit{PExt} : \mathit{Prop} \to 2^{\mathit{Res} \times \mathit{Res}}$, a mapping that assigns an *extension* to each property name; (5) $\mathit{CExt} : \mathit{Class} \to 2^{\mathit{Res}}$ a mapping that assigns a set of resources to every resource denoting a class; (6) $\mathit{Int} : U \to \mathit{Res} \cup \mathit{Prop}$, the *interpretation mapping*, is a mapping that assigns a resource or a property name to each element of U.

Intuitively, a ground triple (s, p, o) in a graph G is true under the interpretation $\mathcal{I}$, if p is *interpreted* as a property name, s and o are *interpreted* as resources, and the interpretation of the pair (s, o) belongs to the extension of the property assigned to p. Formally, we say that $\mathcal{I}$ satisfies the ground triple (s, p, o) if

$\mathit{Int}(p) \in \mathit{Prop}$ and $(\mathit{Int}(s), \mathit{Int}(o)) \in \mathit{PExt}(\mathit{Int}(p))$. An interpretation must also satisfy additional conditions induced by the usage of the RDFS vocabulary. For example, an interpretation satisfying the triple $(c_1, \mathtt{sc}, c_2)$ must interpret c_1 and c_2 as classes of resources, and must assign to c_1 a subset of the set assigned to c_2. More formally, we say that $\mathcal{I}$ satisfies $(c_1, \mathtt{sc}, c_2)$ if $\mathit{Int}(c_1), \mathit{Int}(c_2) \in \mathit{Class}$ and $\mathit{CExt}(c_1) \subseteq \mathit{CExt}(c_2)$.

Blank nodes work as existential variables. Intuitively, a triple (x, p, o) would be true under $\mathcal{I}$, where x is a blank node, if there exists a resource s such that (s, p, o) is true under $\mathcal{I}$. An arbitrary element can be chosen when interpreting a blank node, with the restriction that all the occurrences of the same blank node in an RDF graph must be replaced by the same value. To formally deal with blank nodes, an extension of the interpretation mapping Int is used. Let $A : B \to \mathit{Res}$ be a function between blank nodes and resources. Then $\mathit{Int}_A : UB \to \mathit{Res}$ is defined as the extension of function Int: $\mathit{Int}_A(x) = A(x)$ for $x \in B$, and $\mathit{Int}_A(x) = \mathit{Int}(x)$ for $x \in U$.

We next formalize the notion of *model* for an RDF graph [27,37]. We say that the RDF interpretation $\mathcal{I} = (\mathit{Res}, \mathit{Prop}, \mathit{Class}, \mathit{PExt}, \mathit{CExt}, \mathit{Int})$ is a model of (is an interpretation for) an RDF graph G, denoted by $\mathcal{I} \models G$, if the following conditions hold:

Simple Interpretation:
- there exists a function $A : B \to \mathit{Res}$ such that for each $(s, p, o) \in G$, it holds that $\mathit{Int}(p) \in \mathit{Prop}$ and $(\mathit{Int}_A(s), \mathit{Int}_A(o)) \in \mathit{PExt}(\mathit{Int}(p))$.

Properties and Classes:
- $\mathit{Int}(\mathtt{sp})$, $\mathit{Int}(\mathtt{sc})$, $\mathit{Int}(\mathtt{type})$, $\mathit{Int}(\mathtt{dom})$, $\mathit{Int}(\mathtt{range}) \in \mathit{Prop}$,
- if $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{dom})) \cup \mathit{PExt}(\mathit{Int}(\mathtt{range}))$, then $x \in \mathit{Prop}$ and $y \in \mathit{Class}$.

Sub-property:
- $\mathit{PExt}(\mathit{Int}(\mathtt{sp}))$ is transitive and reflexive over Prop,
- if $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{sp}))$, then $x, y \in \mathit{Prop}$ and $\mathit{PExt}(x) \subseteq \mathit{PExt}(y)$.

Sub-class:
- $\mathit{PExt}(\mathit{Int}(\mathtt{sc}))$ is transitive and reflexive over Class,
- if $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{sc}))$, then $x, y \in \mathit{Class}$ and $\mathit{CExt}(x) \subseteq \mathit{CExt}(y)$.

Typing:
- $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{type}))$ if and only if $y \in \mathit{Class}$ and $x \in \mathit{CExt}(y)$,
- if $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{dom}))$ and $(u, v) \in \mathit{PExt}(x)$, then $u \in \mathit{CExt}(y)$,
- if $(x, y) \in \mathit{PExt}(\mathit{Int}(\mathtt{range}))$ and $(u, v) \in \mathit{PExt}(x)$, then $v \in \mathit{CExt}(y)$.

Example 1. Figure 1 shows an RDF graph storing information about painters. All the triples in the graph are composed by elements in U, except for the triples containing the blank node X. Consider now the interpretation $\mathcal{I} = (\mathit{Res}$, Prop, Class, PExt, CExt, $\mathit{Int})$ defined as follows:

- $\mathit{Res} = \{$Painter, Guayasamin, Cubist, creates, paints, Guernica, Bilbao$\}$.
- $\mathit{Prop} = \{$paints, creates, exhibited_in, $\mathtt{type}, \mathtt{sp}, \mathtt{sc}, \mathtt{dom}, \mathtt{range}\}$.
- $\mathit{Class} = \{$Cubist, Painter$\}$.
- PExt is such that:

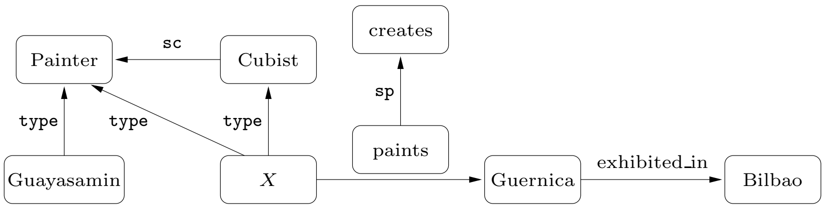


Fig. 1. Example of an RDF graph

  - $PExt$(paints) = $PExt$(creates) = {(Guayasamin, Guernica)},
  - $PExt$(exhibited_in) = {(Guernica, Bilbao)},
  - $PExt$(`type`) = {(Guayasamin, Cubist), (Guayasamin, Painter)},
  - $PExt$(`sp`) = {(paints, create)} $\cup \{(x, x) \mid x \in Prop\}$,
  - $PExt$(`sc`) = {(Cubist, Painter), (Cubist, Cubist), (Painter, Painter)},
  - $PExt$(`dom`) = $PExt$(`range`) = $\emptyset$.
- $CExt$ is such that $CExt$(Cubist) = $CExt$(Painter) = {Guayasamin}.
- Int is the identity mapping over $Res \cup Prop$.

Notice that in our interpretation the sets Res and $Prop$ are subsets of U, but in general, Res and $Prop$ can be arbitrary sets. Let G be the RDF graph of Fig. 1. By considering the function $A : B \to Res$ such that $A(X) =$ Guayasamin, it can be shown that $\mathcal{I} \models G$, that is, $\mathcal{I}$ satisfies all the conditions to be a model of G.

In the interpretation $\mathcal{I}$, we use Guayasamin as a *witness* for the blank node X. Another model of G can use a different witness. For example consider the interpretation $\mathcal{I}' = (Res', Prop, Class, PExt', CExt', Int')$ where:

- $Res' = Res \cup$ {Picasso}.
- $PExt'$ is such that:
  - $PExt'$(paints) = $PExt'$(creates) = {(Picasso, Guernica)},
  - $PExt'$(`type`) = { (Picasso, Cubist), (Picasso, Painter), (Guayasamin, Painter) },
  - $PExt'$ is equal to $PExt$ in every other case.
- $CExt'$ is such that $CExt'$(Cubist) = {Picasso} and $CExt'$(Painter) = {Picasso, Guayasamin}
- Int' is the identity mapping over $Res' \cup Prop$.

It can be shown that interpretation $\mathcal{I}'$ is also a model for G, but this time using Picasso as witness for the blank node X in G. □

2.4 A Deductive System for RDFS

The notion of entailment has shown to be of fundamental importance for many tasks in the database context, and as such it also plays a fundamental role in the context of RDF. Indeed, this notion has been present since the beginning of the Semantic Web initiative. In this section, we study this concept in detail.

Given RDF graphs G_1 and G_2, we say that G_1 *entails* G_2, denoted by $G_1 \models G_2$, if for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models G_1$, it holds that $\mathcal{I} \models G_2$. In [37], the authors showed that this entailment notion between RDF graphs is equivalent to the W3C normative notion of entailment [27], for the fragment of the RDFS vocabulary considered in this paper. In Table 1, we present a deductive system for this notion. This system was given in [37], and is based on a set of rules for $\models$ introduced in [27].

The first rule in Tab. 1 captures the semantics of blank nodes. In every rule (2)-(7), letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{Y}$, stand for *variables* to be replaced by actual terms. More formally, an *instantiation* of a rule (2)-(7) is a replacement of the variables occurring in the triples of the rule by elements of UB, such that all the

Table 1. RDFS inference rules

1. Existential:

$$\frac{G}{G'} \quad \text{for a homomorphism } h : G' \to G$$

2. Subproperty:

(a) $\frac{(\mathcal{A},\texttt{sp},\mathcal{B})\ \ (\mathcal{B},\texttt{sp},\mathcal{C})}{(\mathcal{A},\texttt{sp},\mathcal{C})}$ (b) $\frac{(\mathcal{A},\texttt{sp},\mathcal{B})\ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{X},\mathcal{B},\mathcal{Y})}$

3. Subclass:

(a) $\frac{(\mathcal{A},\texttt{sc},\mathcal{B})\ \ (\mathcal{B},\texttt{sc},\mathcal{C})}{(\mathcal{A},\texttt{sc},\mathcal{C})}$ (b) $\frac{(\mathcal{A},\texttt{sc},\mathcal{B})\ \ (\mathcal{X},\texttt{type},\mathcal{A})}{(\mathcal{X},\texttt{type},\mathcal{B})}$

4. Typing:

(a) $\frac{(\mathcal{A},\texttt{dom},\mathcal{B})\ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{X},\texttt{type},\mathcal{B})}$ (b) $\frac{(\mathcal{A},\texttt{range},\mathcal{B})\ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{Y},\texttt{type},\mathcal{B})}$

5. Implicit Typing:

(a) $\frac{(\mathcal{A},\texttt{dom},\mathcal{B})\ \ (\mathcal{C},\texttt{sp},\mathcal{A})\ \ (\mathcal{X},\mathcal{C},\mathcal{Y})}{(\mathcal{X},\texttt{type},\mathcal{B})}$ (b) $\frac{(\mathcal{A},\texttt{range},\mathcal{B})\ \ (\mathcal{C},\texttt{sp},\mathcal{A})\ \ (\mathcal{X},\mathcal{C},\mathcal{Y})}{(\mathcal{Y},\texttt{type},\mathcal{B})}$

6. Subproperty Reflexivity:

(a) $\frac{(\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{A},\texttt{sp},\mathcal{A})}$ (c) $\frac{}{(p,\texttt{sp},p)}$ for $p \in \{\texttt{sp}, \texttt{sc}, \texttt{dom}, \texttt{range}, \texttt{type}\}$

(b) $\frac{(\mathcal{A},\texttt{sp},\mathcal{B})}{(\mathcal{A},\texttt{sp},\mathcal{A})\ \ (\mathcal{B},\texttt{sp},\mathcal{B})}$ (d) $\frac{(\mathcal{A},p,\mathcal{X})}{(\mathcal{A},\texttt{sp},\mathcal{A})}$ for $p \in \{\texttt{dom}, \texttt{range}\}$

7. Subclass Reflexivity:

(a) $\frac{(\mathcal{A},\texttt{sc},\mathcal{B})}{(\mathcal{A},\texttt{sc},\mathcal{A})\ \ (\mathcal{B},\texttt{sc},\mathcal{B})}$ (b) $\frac{(\mathcal{X},p,\mathcal{A})}{(\mathcal{A},\texttt{sc},\mathcal{A})}$ for $p \in \{\texttt{dom}, \texttt{range}, \texttt{type}\}$

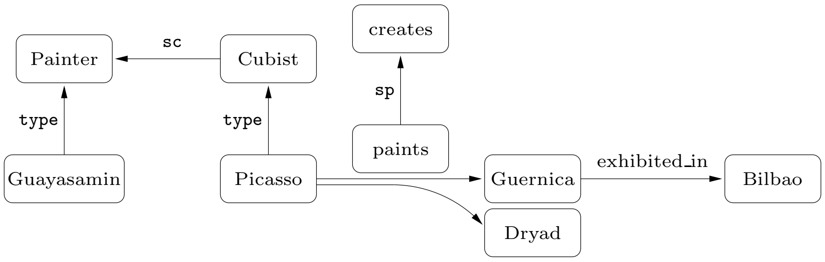


Fig. 2. RDF graph from which we can deduce the graph in Fig. 1

triples obtained after the replacement are well formed RDF triples, that is, not assigning blank nodes to variables in predicate positions.

An *application* of a rule to a graph G is defined as follows. For rule (1), if h is a homomorphism from G' to G, then G' is the result of an application of rule (1) to G. If r is any of the rules (2)-(7), and there is an instantiation $\frac{R}{R'}$ of r such that $R \subseteq G$, then the graph $G' = G \cup R'$ is the result of an application of r to G. We say that a graph G' is *deduced from* G, if G' is obtained from G by successively applying the rules in Tab. 1.

In [37], the authors proved that the set of rules in Tab. 1 is sound and complete for the inference problem for the fragment of RDFS consisting of the reserved words `sc`, `sp`, `range`, `dom` and `type`. That is, it captures the semantics of the normative RDF specification when one focuses on the fragment of the RDFS vocabulary considered in this paper.

Theorem 1 (Soundness and completeness [37]). *Let G and H be RDF graphs, then $G \models H$ iff H is deduced from G by applying rules in Tab. 1.*

It is worth mentioning that the set of rules presented in [27] is not complete for $\models$ (this was pointed out by Marin in [35]). The problem with the system proposed in [27] is that a blank node X can be implicitly used as a property in triples like $(a, \texttt{sp}, X)$, $(X, \texttt{dom}, b)$, and $(X, \texttt{range}, c)$. This problem was solved in [37] by following the approach proposed by Marin [35]. In fact, the rules (5a)-(5b) were added to the system given in [27] to deal with this problem.

Example 2. Let G be the graph in Fig. 1 and G' the graph in Fig. 2. Notice that the triples (Picasso, `type`, Cubist) and (Cubist, `sc`, Painter) belong to G'. Thus, by using rule (3b) we obtain that $G'' = G' \cup \{(\text{Picasso}, \texttt{type}, \text{Painter})\}$ is deduced from G'. Moreover, if we consider a homomorphism h such that $h(X) = \text{Picasso}$, then we have that $h(G) \subseteq h(G'')$, and, thus, applying rule (1) we conclude that G can be deduced from G''. Therefore, the graph G can be deduced from G' by successively applying rules (3b) and (1). Hence, from Theorem 1 we have that every model of G' is also a model of G, i.e. $G' \models G$. □

In [37], the authors showed that the deductive system of Tab. 1 can be simplified by imposing some syntactic restrictions on RDF graphs. The most simple case is obtained when G and H are graphs that do not have blank nodes, and do not mention RDFS vocabulary. In that case, the entailment relation $G \models H$ is reduced to just testing whether $H \subseteq G$. On the other hand, if G and H are RDF graphs that do not mention RDFS vocabulary (but possibly blank nodes), then $G \models H$ if and only if H can be obtained from G by using rule (1), that is, if and only if there exists a homomorphism $h : H \to G$[2]. Another important simplification is obtained if one forbids the presence of *reflexive triples*. A triple t is reflexive if t is of the form $(x, \mathtt{sp}, x)$ or $(x, \mathtt{sc}, x)$ for $x \in UB$. We formalize two of these special cases in the following proposition.

Proposition 1 ([37])

1. *If G and H are RDF graphs that do not mention RDFS vocabulary, then $G \models H$ iff there exists a homomorphism $h : H \to G$.*
2. *If G and H are RDF graphs that have neither blank nodes nor reflexive triples, then $G \models H$ iff H can be deduced from G by using rules* (2)-(4).

In the following sections, we study the fundamental problem of querying RDF data. There is no yet consensus in the Semantic Web community on how to define a query language for RDF that includes all the features of the RDF data model, in particular blank nodes and the RDFS vocabulary. The specification of SPARQL, the standard language for RDF, currently considers RDF data without RDFS vocabulary and with no special semantics for blank nodes. Thus, we study SPARQL in the next sections focusing on ground RDF graphs with no RDFS vocabulary. In Section 6.3, we explore the possibility of having an RDF query language capable of dealing with the special semantics of the RDFS vocabulary.

3 The RDF Query Language SPARQL

In 2004, the RDF Data Access Working Group, part of the W3C Semantic Web Activity, released a first public working draft of a query language for RDF, called SPARQL [45][3]. Since then, SPARQL has been rapidly adopted as the standard for querying Semantic Web data. In January 2008, SPARQL became a W3C Recommendation.

RDF is a directed labeled graph data format and, thus, SPARQL is essentially a graph-matching query language. SPARQL queries are composed by three parts. The *pattern matching part*, which includes several interesting features of pattern matching of graphs, like optional parts, union of patterns, nesting, filtering values of possible matchings, and the possibility of choosing the data source to be matched by a pattern. The *solution modifiers*, which once the output of the pattern has been computed (in the form of a table of values of variables), allow to modify these values applying classical operators like projection, distinct, order and limit. Finally, the *output* of a SPARQL query can be of different types:

[2] Notice that this result is also a corollary of [16].

[3] The name SPARQL is a recursive acronym that stands for *SPARQL Protocol and RDF Query Language.*

yes/no queries, selections of values of the variables which match the patterns, construction of new RDF data from these values, and descriptions of resources.

The definition of a formal semantics for SPARQL has played a key role in the standardization process of this query language. Although taken one by one the features of SPARQL are intuitive and simple to describe and understand, it turns out that the combination of them makes SPARQL into a complex language. Reaching a consensus in the W3C standardization process about a formal semantics for SPARQL was not an easy task. The initial efforts to define SPARQL were driven by use cases, mostly by specifying the expected output for particular example queries. In fact, the interpretations of examples and the exact outcomes of cases not covered in the initial drafts of the SPARQL specification, were a matter of long discussions in the W3C mailing lists. In [40], the authors presented one of the first formalizations of a semantics for a fragment of the language. Currently, the official specification of SPARQL [45], endorsed by the W3C, formalizes a semantics based on [40].

A formalization of a semantics for SPARQL is beneficial for several reasons, including to serve as a tool to identify and derive relations among the constructors that stay hidden in the use cases, identify redundant and contradicting notions, to drive and help the implementation of query engines, and to study the complexity, expressiveness, and further natural database questions like rewriting and optimization. In this section, we present a streamlined version of the core fragment of SPARQL with precise algebraic syntax and a formal compositional semantics based on [40].

One of the delicate issues in the definition of a semantics for SPARQL is the treatment of *optional matching* and incomplete answers. The idea behind optional matching is to allow information to be added if the information is available in the data source, instead of just failing to give an answer whenever some part of the pattern does not match. This feature of optional matching is crucial in Semantic Web applications, and more specifically in RDF data management, where it is assumed that every application have only partial knowledge about the resources being managed. The semantics of SPARQL is formalized by using *partial mappings* between variables in the patterns and actual values in the RDF graph being queried. This formalization allows one to deal with partial answers in a clean way, and is based on the extension of some classical relational algebra operators to work over sets of partial mappings.

A SPARQL query is of the form *head* $\leftarrow$ *body*, where the *body* of the query is a complex RDF graph pattern expression that may include RDF triples with variables, conjunctions, disjunctions, optional parts and constraints over the values of the variables, and the *head* of the query is an expression that indicates how to construct the answer to the query. The evaluation of a query Q against an RDF graph G is done in two steps: the body of Q is matched against G to obtain a set of bindings for the variables in the body, and then using the information on the head of Q, these bindings are processed applying classical relational operators (projection, distinct, etc.) to produce the answer to the query.

It should be noticed that the normative specification of SPARQL [45] is defined over RDF graphs without RDFS vocabulary, and not considering the special semantics of blank nodes. In this section, we work over the same setting.

3.1 Syntax and Semantics of SPARQL Graph Patterns

We first concentrate on the body of SPARQL queries, i.e. in the graph pattern matching facility.

The official syntax of SPARQL [45] considers operators `OPTIONAL`, `UNION`, `FILTER`, and *concatenation* via a point symbol (`.`), to construct graph pattern expressions. The syntax also considers `{ }` to group patterns, and some implicit rules of precedence and association. For example, the point symbol (`.`) has precedence over `OPTIONAL`, and `OPTIONAL` is left associative. In order to avoid ambiguities in the parsing of expressions, we present the syntax of SPARQL graph patterns in a more traditional algebraic formalism, using binary operators AND (`.`), UNION (`UNION`), OPT (`OPTIONAL`), and FILTER (`FILTER`). We fully parenthesize expressions making explicit the precedence and association of operators.

Assume the existence of a set of variables V disjoint from U. A SPARQL graph pattern expression is defined recursively as follows:

1. A tuple from $(U \cup V) \times (U \cup V) \times (U \cup V)$ is a graph pattern (a *triple pattern*).
2. If P_1 and P_2 are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns (*conjunction graph pattern*, *optional graph pattern*, and *union graph pattern*, respectively).
3. If P is a graph pattern and R is a SPARQL *built-in* condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern (a *filter graph pattern*).

A SPARQL *built-in* condition is constructed using elements of the set $U \cup V$ and constants, logical connectives ($\neg$, $\wedge$, $\vee$), inequality symbols ($<$, $\leq$, $\geq$, $>$), the equality symbol ($=$), unary predicates like bound, isBlank, and isIRI, plus other features (see [45] for a complete list). In this paper, we restrict to the fragment where the built-in condition is a Boolean combination of terms constructed by using $=$ and bound, that is:

1. If $?X, ?Y \in V$ and $c \in U$, then $\text{bound}(?X)$, $?X = c$ and $?X =?Y$ are built-in conditions.
2. If R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions.

Let P be a SPARQL graph pattern. In the rest of the paper, we use $\text{var}(P)$ to denote the set of variables occurring in P. In particular, if t is a triple pattern, then $\text{var}(t)$ denotes the set of variables occurring in the components of t. Similarly, for a built-in condition R, we use $\text{var}(R)$ to denote the set of variables occurring in R.

To define the semantics of SPARQL graph pattern expressions, we need to introduce some terminology. A *mapping* μ from V to U is a partial function

$\mu : V \to U$. Abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. The domain of μ, denoted by $\mathrm{dom}(\mu)$, is the subset of V where μ is defined. Two mappings μ_1 and μ_2 are *compatible* when for all $?X \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping. Intuitively, μ_1 and μ_2 are compatibles if μ_1 *can be extended* with μ_2 to obtain a new mapping, and vice versa. Note that two mappings with disjoint domains are always compatible, and that the empty mapping $\mu_\emptyset$ (i.e. the mapping with empty domain) is compatible with any other mapping.

Let Ω_1 and Ω_2 be sets of mappings. We define the join of, the union of and the difference between Ω_1 and Ω_2 as [40]:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\},\\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\},\\
\Omega_1 \smallsetminus \Omega_2 &= \{\mu \in \Omega_1 \mid \text{ for all } \mu' \in \Omega_2,\ \mu \text{ and } \mu' \text{ are not compatible}\}.
\end{aligned}$$

Based on the previous operators, we define the left outer-join as:

$$\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2).$$

Intuitively, $\Omega_1 \bowtie \Omega_2$ is the set of mappings that result from extending mappings in Ω_1 with their compatible mappings in Ω_2, and $\Omega_1 \smallsetminus \Omega_2$ is the set of mappings in Ω_1 that cannot be extended with any mapping in Ω_2. The operation $\Omega_1 \cup \Omega_2$ is the usual set theoretical union. A mapping μ is in $\Omega_1 ⟕ \Omega_2$ if it is the extension of a mapping of Ω_1 with a compatible mapping of Ω_2, or if it belongs to Ω_1 and cannot be extended with any mapping of Ω_2. These operations resemble relational algebra operations over sets of mappings (partial functions) [52].

We are ready to define the semantics of graph pattern expressions as a function $[\![\cdot]\!]_G$ which takes a pattern expression and returns a set of mappings. We follow the approach in [23] defining the semantics as the set of mappings that matches the graph G. For the sake of readability, the semantics of filter expressions is presented in a separate definition.

Definition 2. *The* evaluation *of a graph pattern P over an RDF graph G, denoted by $[\![P]\!]_G$, is defined recursively as follows:*

1. *if P is a triple pattern t, then $[\![P]\!]_G = \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t)$ and $\mu(t) \in G\}$.*
2. *if P is $(P_1$ AND $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$.*
3. *if P is $(P_1$ OPT $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G ⟕ [\![P_2]\!]_G$.*
4. *if P is $(P_1$ UNION $P_2)$, then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$.*

The idea behind the OPT operator is to allow for *optional matching* of patterns. Consider pattern expression $(P_1$ OPT $P_2)$ and let μ_1 be a mapping in $[\![P_1]\!]_G$. If there exists a mapping $\mu_2 \in [\![P_2]\!]_G$ such that μ_1 and μ_2 are compatible, then $\mu_1 \cup \mu_2$ belongs to $[\![(P_1$ OPT $P_2)]\!]_G$. But if no such a mapping μ_2 exists, then μ_1 belongs to $[\![(P_1$ OPT $P_2)]\!]_G$. Thus, operator OPT allows information to be added to a mapping μ if the information is available, instead of just rejecting μ whenever some part of the pattern does not match.

The semantics of filter expressions goes as follows. Given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if:

1. R is bound$(?X)$ and $?X \in \text{dom}(\mu)$;
2. R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
3. R is $?X = ?Y$, $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
4. R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\mu \models R_1$;
5. R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions, and $\mu \models R_1$ or $\mu \models R_2$;
6. R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\mu \models R_1$ and $\mu \models R_2$.

Definition 3. *Given an RDF graph G and a filter expression $(P \text{ FILTER } R)$,*

$$[\![(P \text{ FILTER } R)]\!]_G = \{\mu \in [\![P]\!]_G \mid \mu \models R\}.$$

In the normative semantics of SPARQL [45], there is an additional feature of graph patterns that allows to query several different RDF graphs with a single pattern. This is accomplished with the GRAPH operator that allows to dynamically change the graph being used in the evaluation of a pattern. For the sake of readability, we do not include here the GRAPH operator. We refer the reader to [42] for a formalization of SPARQL graph patterns including GRAPH, and to [9] for some tutorial material.

In the rest of the paper, we usually represent sets of mappings as tables where each row represents a mapping in the set. We label every row with the name of a mapping, and every column with the name of a variable. If a mapping is not defined for some variable, then we simply leave empty the corresponding position. For instance, the table

	$?X$	$?Y$	$?Z$	$?V$	$?W$
μ_1 :	a	b			
μ_2 :		c			d
μ_3 :			e		

represents the set $\Omega = \{\mu_1, \mu_2, \mu_3\}$ where

- $\text{dom}(\mu_1) = \{?X, ?Y\}$, $\mu_1(?X) = a$, and $\mu_1(?Y) = b$,
- $\text{dom}(\mu_2) = \{?Y, ?W\}$, $\mu_2(?Y) = c$, and $\mu_2(?W) = d$,
- $\text{dom}(\mu_3) = \{?Z\}$, and $\mu_3(?Z) = e$.

Sometimes we use notation $\{\{?X \to a, ?Y \to b\}, \{?Y \to c, ?W \to d\}, \{?Z \to e\}\}$ for a set of mappings as the one above.

Example 3. Consider an RDF graph G storing information about professors in a university:

$$G = \{ \begin{array}{llllll} (B_1, \text{name}, & \text{paul}), & & (B_1, \text{phone}, & \text{777-3426}), & \\ (B_2, \text{name}, & \text{john}), & & (B_2, \text{email}, & \text{john@acd.edu}), & \\ (B_3, \text{name}, & \text{george}), & & (B_3, \text{webPage}, & \text{www.george.edu}), & \\ (B_4, \text{name}, & \text{ringo}), & & (B_4, \text{email}, & \text{ringo@acd.edu}), & \\ (B_4, \text{webPage}, & \text{www.starr.edu}), & & (B_4, \text{phone}, & \text{888-4537}) & \} \end{array}$$

The following are graph pattern expressions and their evaluations over G:

- $P_1 = ((?A, \text{email}, ?E) \text{ AND } (?A, \text{webPage}, ?W))$. Then

$\llbracket P_1 \rrbracket_G =$

	$?A$	$?E$	$?W$
μ_1 :	B_4	ringo@acd.edu	www.starr.edu

- $P_2 = ((?A, \text{email}, ?E) \text{ OPT } (?A, \text{webPage}, ?W))$. Then

$\llbracket P_2 \rrbracket_G =$

	$?A$	$?E$	$?W$
μ_1 :	B_2	john@acd.edu	
μ_2 :	B_4	ringo@acd.edu	www.starr.edu

- $P_3 = (((?A, \text{name}, ?N) \text{ OPT } (?A, \text{email}, ?E)) \text{ OPT } (?A, \text{webPage}, ?W))$. Then

$\llbracket P_3 \rrbracket_G =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		www.george.edu
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

- $P_4 = ((?A, \text{name}, ?N) \text{ OPT } ((?A, \text{email}, ?E) \text{ OPT } (?A, \text{webPage}, ?W)))$. Then

$\llbracket P_4 \rrbracket_G =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

  Notice the difference between $\llbracket P_2 \rrbracket_G$ and $\llbracket P_3 \rrbracket_G$. These two examples show that $\llbracket ((A \text{ OPT } B) \text{ OPT } C) \rrbracket_G \neq \llbracket (A \text{ OPT } (B \text{ OPT } C)) \rrbracket_G$ in general.
- $P_5 = ((?A, \text{name}, ?N) \text{ AND } ((?A, \text{email}, ?E) \text{ UNION } (?A, \text{webPage}, ?W)))$. Then

$\llbracket P_5 \rrbracket_G =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_2	john	john@acd.edu	
μ_2 :	B_3	george		www.george.edu
μ_3 :	B_4	ringo	ringo@acd.edu	
μ_4 :	B_4	ringo		www.starr.edu

- $P_6 = (((?A, \text{name}, ?N) \text{ OPT } (?A, \text{phone}, ?P)) \text{ FILTER } ?N = \text{paul})$. Then

$\llbracket P_6 \rrbracket_G =$

	$?A$	$?N$	$?P$
μ_1 :	B_1	paul	777-3426

□

Simple algebraic properties. We say that two graph patterns P_1 and P_2 are *equivalent*, denoted by $P_1 \equiv P_2$, if $[\![P_1]\!]_G = [\![P_2]\!]_G$ for every RDF graph G. The following simple lemma states some simple algebraic properties of AND and UNION operators. These properties are direct consequence of the semantics of AND and UNION, both based on set-theoretical union.

Lemma 1 ([40]). *The operators* AND *and* UNION *are associative and commutative and the operator* AND *distribute over* UNION. *That is, if P_1, P_2 and P_3 are graph patterns, then it holds that:*

– $(P_1 \text{ AND } P_2) \equiv (P_2 \text{ AND } P_1)$
– $(P_1 \text{ UNION } P_2) \equiv (P_2 \text{ UNION } P_1)$
– $(P_1 \text{ AND } (P_2 \text{ AND } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ AND } P_3)$
– $(P_1 \text{ UNION } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ UNION } P_2) \text{ UNION } P_3)$
– $(P_1 \text{ AND } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3))$

The above lemma permits us to avoid parenthesis when writing sequences of either AND operators or UNION operators. This is consistent with the definitions of *Group Graph Pattern* and *Union Graph Pattern* in [45]. We use Lemma 1 to simplify the notation in the following sections.

3.2 Query Result Forms

The normative specification of SPARQL [45] considers four *query forms*. These query forms use the mappings obtained after the evaluation of a graph pattern to construct result sets or RDF graphs. The query forms are: (1) SELECT, that performs a *projection* over a set of variables in the evaluation of a graph pattern, (2) CONSTRUCT, that returns an RDF graph constructed by substituting variables in a *template*, (3) ASK, that returns a truth value indicating whether the evaluation of a graph pattern produces at least one mapping, and (4) DESCRIBE, that returns an RDF graph that describes the resources found. In this paper, we only consider the SELECT query form. We refer the reader to [42] for a formalization of the remaining query forms.

Given a mapping $\mu : V \to U$ and a set of variables $W \subseteq V$, the *restriction* of μ to W, denoted by $\mu_{|_W}$, is a mapping such that $\text{dom}(\mu_{|_W}) = \text{dom}(\mu) \cap W$ and $\mu_{|_W}(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu) \cap W$.

Definition 4. *A* SPARQL SELECT query *is a tuple (W, P), where P is a graph pattern and W is a set of variables such that $W \subseteq \text{var}(P)$. The* answer *of (W, P) over an RDF graph G, denoted by $[\![(W, P)]\!]_G$, is the set of mappings:*

$$[\![(W, P)]\!]_G = \{\mu_{|_W} \mid \mu \in [\![P]\!]_G\}.$$

Example 4. Consider the RDF graph G and the graph pattern P_3 in Example 3. Then we have that:

$$[\![(\{?N,?E\},P_3)]\!]_G = \begin{array}{r|c|c|} \cline{2-3} & ?N & ?E \\ \cline{2-3} \mu_1: & \text{paul} & \\ \mu_2: & \text{john} & \text{john@acd.edu} \\ \mu_3: & \text{george} & \\ \mu_4: & \text{ringo} & \text{ringo@acd.edu} \\ \cline{2-3} \end{array}$$

$\square$

In the following sections, we study some fundamental issues regarding the query language SPARQL. The first of that issues is the complexity of the evaluation problem for SPARQL. In Section 4, we focus on studying the complexity of the evaluation problem for SPARQL graph patterns. Then in Section 5, we consider SPARQL SELECT queries to compare the expressive powers of SPARQL and the Relational Algebra.

4 Complexity and Optimization of SPARQL

A fundamental issue in every query language is the complexity of query evaluation and, in particular, what is the influence of each component of the language in this complexity.

In this section, we present a thorough study of the complexity of the evaluation of SPARQL graph patterns based on [40]. In this study, we consider several fragments of SPARQL built incrementally, and present complexity results for each such fragment. Among other results, we show that the complexity of the evaluation problem for general SPARQL graph patterns is PSPACE-complete [40], and that this high complexity is obtained as a consequence of unlimited use of nested optional parts.

Given the high complexity of the evaluation problem for general SPARQL graph patterns, an important question is whether one can find interesting classes of patterns where the query evaluation problem can be solved more efficiently. In [40,41], the authors identified a large class of patterns with the previous characteristic that is defined by a simple and natural syntactic restriction. This class is obtained by forbidding a special form of interaction between variables appearing in optional parts. Patterns satisfying this condition are called *well-designed* [40,41]. Well-designed patterns form a natural fragment of SPARQL that is very common in practice, and has several interesting features. On the one hand, the complexity of the evaluation problem for well-designed patterns is considerably lower, namely coNP-complete. On the other hand, the property of being well designed has important consequences for the optimization of SPARQL queries. We present some rewriting rules for well-designed patterns whose application may have a considerable impact in the cost of evaluating SPARQL queries, and prove the existence of a normal form for well-designed patterns based on the application of these rewriting rules.

4.1 Complexity of Evaluating Graph Pattern Expressions

In this section, we review some the results in the literature regarding the complexity of evaluating SPARQL graph pattern expressions. The first study about this problem was published in [40], and some refinements of the complexity results of [40] were presented in [47]. This section focuses on the complexity results proved in these two papers.

As is customary when studying the complexity of the evaluation problem for a query language [51], we consider its associated decision problem. We denote this problem by EVALUATION and we define it as follows:

INPUT	: An RDF graph G, a graph pattern P and a mapping μ.
QUESTION	: Is $\mu \in [\![P]\!]_G$?

It is important to notice that the evaluation problem that we study considers the mapping as part of the input. That is, we study the complexity by measuring how difficult it is to verify whether a given mapping is a solution for a pattern evaluated over an RDF graph. This is the standard *decision* problem considered when studying the complexity of a query language [51], as opposed to the *computation* problem of actually listing the set of solutions (finding all the mappings). To focus on the associated decision problem allows us to obtain a fine grained analysis of the complexity of the evaluation problem, classifying the complexity for different fragments of SPARQL in terms of standard complexity classes. Also notice that the pattern and the graph are both input for EVALUATION. Thus, we study the *combined complexity* of the query language [51].

We start this study by considering the fragment consisting of graph pattern expressions constructed by using only AND and FILTER operators. This simple fragment is interesting as it does not use the two most complicated operators in SPARQL, namely UNION and OPT. Given an RDF graph G, a graph pattern P in this fragment and a mapping μ, it is possible to efficiently check whether $\mu \in [\![P]\!]_G$ by using the following simple algorithm [40]. First, for each triple t in P, verify whether $\mu(t) \in G$. If this is not the case, then return *false*. Otherwise, by using a bottom-up approach, verify whether the expression generated by instantiating the variables in P according to μ satisfies the FILTER conditions in P. If this is the case, then return *true*, else return *false*.

Theorem 2. EVALUATION *can be solved in time* $O(|P| \cdot |D|)$ *for graph pattern expressions constructed by using only* AND *and* FILTER *operators.*

We continue this study by adding the UNION operator to the AND-FILTER fragment. It is important to notice that the inclusion of UNION in SPARQL is one of the most controversial issues in the definition of this language. The following theorem proved in [40], shows that the inclusion of the UNION operator makes the evaluation problem for SPARQL considerably harder.

Theorem 3 ([40]). EVALUATION *is NP-complete for graph pattern expressions constructed by using only* AND, FILTER *and* UNION *operators.*

In [47], the authors strengthen the above result by showing that the complexity of evaluating graph pattern expressions constructed by using only AND and UNION operators is already NP-hard. Thus, we have the following result.

Theorem 4 ([47]). EVALUATION *is NP-complete for graph pattern expressions constructed by using only* AND *and* UNION *operators.*

We now consider the OPT operator, which is the most involved operator in graph pattern expressions and, definitively, the most difficult to define. The following theorem proved in [40] shows that when considering all the operators in SPARQL graph patterns, the evaluation problem becomes considerably harder.

Theorem 5 ([40]). EVALUATION *is PSPACE-complete.*

To prove the PSPACE-hardness of EVALUATION, the authors show in [40] how to reduce in polynomial time the quantified boolean formula problem (QBF) to EVALUATION. An instance of QBF is a quantified propositional formula φ of the form:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_m \exists y_m \, \psi,$$

where ψ is a quantifier-free formula of the form $C_1 \wedge \cdots \wedge C_n$, with each C_i $(i \in \{1, \ldots, n\})$ being a disjunction of literals, that is, a disjunction of propositional variables x_i and y_j, and negations of propositional variables. Then the problem is to verify whether φ is valid. It is known that QBF is PSPACE-complete [22]. In the encoding presented in [40], the authors use a fixed RDF graph G and a fixed mapping μ. Then they encode formula φ with a pattern P_φ that uses nested OPT operators to encode the *quantifier alternation* of φ, and a graph pattern without OPT to encode the satisfiability of formula ψ. By using a similar idea, it is shown in [47] how to encode formulas φ and ψ by using only the OPT operator, thus strengthening Theorem 5.

Theorem 6 ([47]). EVALUATION *is PSPACE-complete for graph pattern expressions constructed by using only the* OPT *operator.*

When verifying whether $\mu \in [\![P]\!]_G$, it is natural to assume that the size of P is considerably smaller than the size of G. This assumption is very common when studying the complexity of a query language. In fact, it is named *data complexity* in the database literature [51], and it is defined as the complexity of the evaluation problem for a fixed query. More precisely, for the case of SPARQL, given a graph pattern expression P, the evaluation problem for P, denoted by EVALUATION(P), has as input an RDF graph G and a mapping μ, and the problem is to verify whether $\mu \in [\![P]\!]_G$.

Theorem 7 ([40]). EVALUATION(P) *is in LOGSPACE for every graph pattern expression P.*

An important question is whether one can find interesting classes of graph patterns, constructed by imposing simple and natural syntactic restrictions, such that one can obtain lower complexity bounds for the evaluation problem on that classes. In the following section, we introduce a first such restriction.

4.2 A Simple Normal Form for Graph Patterns

We say that a pattern P is UNION-free if P is constructed by using only operators AND, OPT and FILTER. In [40], the authors proved the following normal-form result.

Proposition 2 ([40]). *Every graph pattern P is equivalent to a pattern of the form:*

$$(P_1 \quad \text{UNION} \quad P_2 \quad \text{UNION} \quad P_3 \quad \text{UNION} \quad \cdots \quad \text{UNION} \quad P_n), \qquad (1)$$

where each P_i $(1 \leq i \leq n)$ is UNION*-free.*

Notice that we omit the parenthesis in the expression (1) given the associativity of UNION. We say that a graph pattern is in UNION *normal form* if the pattern is in the form (1).[4]

The following result shows that for graph patterns in UNION normal form that do not use the OPT operator, the evaluation problem can be solved efficiently. It is a direct consequence of Theorem 2.

Corollary 1. Evaluation *can be solved in time $O(|P| \cdot |G|)$ for graph patterns in* UNION *normal form constructed by using only* AND, FILTER, *and* UNION *operators.*

We have managed to lower the complexity of the AND-FILTER-UNION fragment by imposing a simple normal form. However, Theorem 6 implies that when the OPT operator is allowed in graph patterns, the complexity of the evaluation problem is PSPACE-hard even if we restrict to patterns in UNION normal form. In the following section, we introduce a simple and natural syntactic condition that patterns usually satisfy in practice. Under this condition, the complexity of the evaluation of graph patterns in UNION normal form is lower even if the OPT operator is allowed.

4.3 Well-Designed Graph Patterns

The exact semantics of graph pattern expressions has been extensively discussed on the mailing list of the W3C. One of the most delicate issues in the definition of a semantics for graph pattern expressions is the semantics of the OPT operator. As we have mentioned before, the idea behind the OPT operator is to allow for *optional matching* of patterns, that is, to allow information to be added if it is available, instead of just rejecting whenever some part of a pattern does not

[4] In the conference version of [40], the proof of the existence of a UNION normal form used the equivalence $(P_1$ OPT $(P_2$ UNION $P_3)) \equiv ((P_1$ OPT $P_2)$ UNION $(P_1$ OPT $P_3))$ (see Proposition 1 in [40]). Unfortunately, this rule does not hold in general [47]. In the *errata* of [40] (that can be downloaded from `http://www.ing.puc.cl/~marenas/publications/errata-iswc06.pdf`), the authors provide a proof of Proposition 2 without using this rule.

match. However, this intuition fails in some simple, but unnatural, examples. For instance, consider the graph pattern:

$$P = ((?X, \text{name}, \text{john}) \text{ OPT } ((?Y, \text{name}, \text{mick}) \text{ OPT } (?X, \text{email}, ?Z))). \quad (2)$$

What is unnatural about graph pattern P is the fact that $(?X, \text{email}, ?Z)$ is giving optional information for $(?X, \text{name}, \text{john})$, but in P appears as giving optional information for $(?Y, \text{name}, \text{mick})$. For example, $(B_2, \text{name}, \text{john})$ and $(B_2, \text{email}, \text{john@ac.edu})$ are triples in the graph G of Example 3, but the evaluation of P results in the set $\{\{?X \to B_2\}\}$ (since $[\![(?Y, \text{name}, \text{mick})]\!]_G = \emptyset$) without giving information about the email of john.

A careful examination of the examples that produce conflicts reveals a common pattern: A graph pattern P mentions an expression $P' = (P_1 \text{ OPT } P_2)$ and a variable $?X$ occurring both inside P_2 and outside P' but not occurring in P_1. In general, graph pattern expressions satisfying this condition are not natural.

In [40], the authors considered a special class of patterns that they called *well-designed patterns*, obtained by forbidding the form of interaction between variables appearing in optional parts discussed above. To present the formal definition of well-designed patterns, we need to introduce some terminology. We say that a graph pattern Q is *safe* if for every sub-pattern $(P \text{ FILTER } R)$ of Q, it holds that $\text{var}(R) \subseteq \text{var}(P)$. This safety condition is a usual restriction in many database query languages.

Definition 5 ([40]). *A* UNION*-free graph pattern P is* well designed *if P is safe and, for every sub-pattern $P' = (P_1 \text{ OPT } P_2)$ of P and for every variable $?X$ occurring in P, the following condition holds:*

if $?X$ occurs both inside P_2 and outside P', then it also occurs in P_1.

For instance, pattern (2) above is not well designed. One can extend Definition 5 to patterns in UNION normal form; a pattern $(P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n)$ is well designed if every P_i $(1 \leq i \leq n)$ is a UNION-free well-designed graph pattern.

It should be noticed that to prove the PSPACE lower bound of Theorem 5, it is used in [40] a graph pattern that is not well designed. Thus, an immediate question is whether the complexity of evaluating well-designed graph pattern expressions is lower than in the general case. In [41] (the extended version of [40]), the authors showed that this is indeed the case, in fact, they proved a coNP upper bound for the case of well-designed graph patterns. In [40,41], the authors also considered the problem of optimizing well-designed graph patterns. Since the beginning of the relational model, several techniques for optimizing the evaluation of relational algebra expressions have been developed. In fact, one of the reasons why relational algebra is so extensively used to implement SQL is the existence of simple reordering and optimization rules for this language. Unfortunately, the development of this type of rules for SPARQL is limited by the presence of the OPT operator. However, it was shown in [40,41] that well-designed patterns are suitable for reordering and optimization, demonstrating the significance of this class of queries from the practical point of view. In the rest of this section, we review some of the results in [40,41] regarding well-designed patterns.

We note first that the property of being well-designed can be checked efficiently by a straightforward procedure. Let P be a pattern. Then for every sub-pattern P' of P of the form $(P_1 \text{ OPT } P_2)$, we construct three sets: sets V_{P_1} and V_{P_2}, containing the variables occurring in P_1 and P_2, respectively, and set $O_{P'}$ containing the variables that occur *outside* P'. To construct V_{P_1}, we collect variables by making a bottom-up traversal of the sub-patterns of P_1. We repeat this procedure in P_2 to construct V_{P_2}. To construct $O_{P'}$, we make a bottom-up traversal of the entire pattern P, but not taking into consideration P'. Having these three sets, we check whether $V_{P_2} \cap O_{P'} \subseteq V_{P_1}$, that is, we check whether every variable that occurs inside P_2 and outside P' also occurs inside P_1, which is exactly the well-designed condition. We must repeat this test for every OPT sub-pattern of P. Notice that the test for every OPT sub-pattern takes linear time in the size of P, and then, the entire process takes time proportional to the size of P times the number of OPT sub-patterns of P. We can then state the following proposition:

Proposition 3 ([41]). *Testing if a pattern P is well designed can be done in time $O(|P|^2)$.*

4.4 Complexity of Evaluating Well-Designed Patterns

Intuitively, if we *delete* some optional parts of a pattern P to obtain a new pattern P', the mappings in the evaluation of P' over a graph G could not be more informative than the mappings in the evaluation of P over G. That is, the optional matchings of a pattern must only serve to *extend* solutions with new information, but not to reject solutions if some information is not provided. In [41], the authors showed that the intuition is indeed correct for the case of well-designed graph patterns. In this section, we present the formalization of this intuition given in [41], and use it to develop a characterization of the evaluation of well-designed graph patterns.

We say that a mapping μ is *subsumed* by a mapping μ', denoted by $\mu \sqsubseteq \mu'$, if μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$. That is, μ is subsumed by μ' if μ agrees with μ' in every variable for which μ is defined. For sets of mappings Ω and Ω', we write $\Omega \sqsubseteq \Omega'$ if for every mapping $\mu \in \Omega$, there exists a mapping $\mu' \in \Omega'$ such that $\mu \sqsubseteq \mu'$.

We say that a pattern P' is a *reduction* of a pattern P, if P' can be obtained from P by replacing a sub-formula $(P_1 \text{ OPT } P_2)$ of P by P_1, that is, if P' is obtained by deleting some optional part of P. For example,

$$P' = (t_1 \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)))$$

is a reduction of

$$P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)))$$

since P' can be obtained from P by replacing $(t_1 \text{ OPT } t_2)$ by t_1. The reflexive and transitive closure of the reduction relation is denoted by $\trianglelefteq$. Thus, for example, if $P'' = (t_1 \text{ AND } t_2)$, then $P'' \trianglelefteq P$ since P'' is a reduction of P' and P' is a reduction of P. We note that if $P' \trianglelefteq P$ and P is well designed, then P' is well designed.

We can now state the result that formalizes the intuition mentioned at the beginning of this section.

Lemma 2 ([41]). *Let P be a* UNION*-free well-designed graph pattern, and P' a pattern such that $P' \trianglelefteq P$. Then $[\![P']\!]_G \sqsubseteq [\![P]\!]_G$ for every graph G.*

It should be noticed that the property stated in Lemma 2 does not hold for patterns that are not well designed. For example, consider a graph $G = \{(1, \mathtt{a}, 1), (2, \mathtt{a}, 2), (3, \mathtt{a}, 3)\}$ and non well-designed pattern:

$$P = ((?X, \mathtt{a}, 1) \text{ OPT } ((?Y, \mathtt{a}, 2) \text{ OPT } (?X, \mathtt{a}, 3))).$$

The evaluation of P results in the set $\{\{?X \to 1\}\}$. By deleting the optional part $(?X, \mathtt{a}, 3)$ of P, we obtain the reduction $P' = ((?X, \mathtt{a}, 1) \text{ AND } (?Y, \mathtt{a}, 2))$ of P. The evaluation of P' results in the set $\{\{?X \to 1, ?Y \to 2\}\}$. Thus, we have that $[\![P']\!]_G \not\sqsubseteq [\![P]\!]_G$.

We have mentioned that, when evaluating an optional part of a pattern, one is trying to extend mappings with optional information. Another intuition behind the OPT operator is that, when a pattern has several optional parts, one wants to extend the solutions *as much as possible*, that is, one does not want to lose information when the information is present. We formalize this intuition with the notion of *partial solution* for a pattern. Informally, a partial solution for a pattern P is a mapping that is an *exact match* for some P' such that $P' \trianglelefteq P$. We show then, in Proposition 4, that the evaluation of a well-designed graph pattern P is exactly the set of *maximal partial solutions* for P w.r.t. $\sqsubseteq$, that is, the solutions that retrieve as much information as possible. This proposition gives an alternative characterization of the evaluation of well-designed graph patterns.

Given a pattern P, define $\text{and}(P)$ to be the pattern obtained from P by replacing every OPT operator in P by an AND operator. For example, if P is the pattern:

$$P = ((t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4))),$$

then we have that:

$$\text{and}(P) = ((t_1 \text{ AND } t_2) \text{ AND } (t_2 \text{ AND } (t_3 \text{ AND } t_4))).$$

Notice that, by the semantics of the OPT operator, for every (not necessarily well designed) pattern P and every graph G, we have that $[\![\text{and}(P)]\!]_G \subseteq [\![P]\!]_G$.

A mapping μ is a *partial solution* for a pattern P over a graph G if $\mu \in [\![\text{and}(P')]\!]_G$, for some $P' \trianglelefteq P$. Partial solutions and the notion of subsumption of mappings give the following characterization of the evaluation of well-designed graph patterns.

Proposition 4 ([41]). *Given a* UNION*-free well-designed graph pattern P, a graph G, and a mapping μ, we have that $\mu \in [\![P]\!]_G$ if and only if μ is a maximal (w.r.t. $\sqsubseteq$) partial solution for P over G.*

In [41], the authors use this characterization to prove that the complexity of the evaluation problem for well-designed patterns is lower than for general patterns.

Theorem 8 ([41]). EVALUATION *is coNP-complete for the case of* UNION*-free well-designed graph pattern expressions.*

The characterization of the evaluation of well-designed graph patterns in Proposition 4 can be extended to patterns in UNION normal form. For a well-designed pattern $P = (P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n)$ in UNION normal form, a mapping μ, and a graph G, it holds that $\mu \in [\![P]\!]_G$ if and only if μ is a maximal partial solution (w.r.t. $\sqsubseteq$) for some P_i $(1 \leq i \leq n)$. Then the evaluation problem for well-designed patterns in UNION normal form is still in coNP.

Corollary 2 ([41]). EVALUATION *is coNP-complete for well-designed graph pattern expressions in* UNION *normal form.*

4.5 Optimization of Well-Designed Patterns

Due to the evident similarity between certain operators of SPARQL and relational algebra, a natural question is whether the classical results of normal forms and optimization for relational algebra are applicable in the SPARQL context. The answer is not straightforward, at least for the case of optional patterns and its relational counterpart, the left outer join. The classical results about outer-join query reordering and optimization by Galindo-Legaria and Rosenthal [21] are not directly applicable in the SPARQL context, as they assume constraints on the relational queries that are rarely satisfied in SPARQL. The first, and most problematic issue, is the assumption on predicates used for joining/outer-joining relations to be *null-rejecting* [21]. A predicate p is null-rejecting if it evaluates to *false* (or *undefined*) whenever a null value is used in p. In SPARQL, those predicates are implicit in the variables that graph patterns share and, by the definition of compatible mappings, they are never *null-rejecting*. In fact, people who have developed algorithms for translating SPARQL queries into relational algebra and SQL queries (e.g. [20]) have used `NULL` to represent unbound variables, `IS NULL` in predicates for joining/outer-joining, and `COALESCE` for merging the values of different columns into a single column. These features are explicitly prohibited in [21] since they may imply a violation of the null-rejecting requirement.

Since the application of classical results in relational query optimization is not straightforward, it would be desirable to develop specific techniques in the SPARQL context. In [40], the authors proved that the property of being well designed has important consequences for the study of normalization and optimization for SPARQL.

Proposition 5 ([40]). *Let P_1, P_2 and P_3 be graph pattern expressions and R a built-in condition. Consider the rewriting rules:*

$$((P_1 \text{ OPT } P_2) \text{ FILTER } R) \longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2), \quad (3)$$
$$(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) \longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3), \quad (4)$$
$$((P_1 \text{ OPT } P_2) \text{ AND } P_3) \longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2). \quad (5)$$

Let P be a UNION*-free well-designed pattern, and assume that P' is a pattern obtained from P by applying either Rule (3), or Rule (4), or Rule (5). Then P' is a* UNION*-free well-designed pattern equivalent to P.*

It is worth mentioning that the previous rules are not applicable to non well-designed graph patterns. For example, consider the graph $G = \{(1, \mathtt{a}, 1), (2, \mathtt{a}, 2), (3, \mathtt{a}, 3)\}$ and non well-designed pattern:

$$P = ((?X, \mathtt{a}, 1) \text{ AND } ((?Y, \mathtt{a}, 2) \text{ OPT } (?X, \mathtt{a}, 3))).$$

The evaluation of P results in the empty set of mappings. If we apply rule (4) to P, we obtain pattern $P' = (((?X, \mathtt{a}, 1) \text{ AND } (?Y, \mathtt{a}, 2)) \text{ OPT } (?X, \mathtt{a}, 3))$. The evaluation of P' results in the set $\{\{?X \to 1, ?Y \to 2\}\}$ and, thus, we have that $[\![P]\!]_G \neq [\![P']\!]_G$.

We say that a UNION-free graph pattern P is in OPT *normal form* if either: (1) P is constructed by using only the AND and FILTER operators, or (2) $P = (O_1 \text{ OPT } O_2)$, with O_1 and O_2 patterns in OPT normal form. For example, consider a pattern P:

$$\Big[\Big(((t_1 \text{ AND } t_2) \text{ FILTER } R_1) \\ \text{OPT } (t_3 \text{ OPT } ((t_4 \text{ FILTER } R_2) \text{ AND } t_5))\Big) \text{ OPT } \Big(t_6 \text{ FILTER } R_3\Big)\Big],$$

where every t_i is a triple pattern, and every R_j is a built-in condition. Then P is in OPT normal form. The following theorem shows that for every well-designed graph pattern, an equivalent pattern in OPT normal form can be efficiently obtained.

Theorem 9 ([41]). *For every* UNION*-free well-designed pattern P, an equivalent pattern in* OPT *normal form can be obtained after $O(|P|^2)$ applications of Rules (3)-(5).*

The application of Rules (3)-(5) may have a considerable impact in the cost of evaluating graph patterns. One can measure this impact by analyzing the intermediate sizes of the sets of mappings produced when evaluating a pattern. By the semantics of the OPT operator, when evaluating an expression of the form $(P_1 \text{ OPT } P_2)$ over a graph G, the number of mappings obtained is at least the number of mappings obtained when evaluating P_1 over D. That is, the application of the OPT operator never implies a reduction in the size of the intermediate results in the evaluation of a graph pattern expression. In contrast, it is clear that operators AND and FILTER may imply a reduction in the size of intermediate results. Thus, for optimization purposes, it would be convenient to perform all the AND and FILTER operations first, delaying the OPT operations to the last step of the evaluation. A pattern in OPT normal form has its operators ordered in a way that, the bottom-up evaluation of the pattern follows exactly this strategy: AND and FILTER operations are executed prior to the execution of the OPT operations.

5 On the Expressiveness of SPARQL

Determining the expressive power of a query language is crucial for understanding its capabilities, that is, what types of queries a user can pose in this language, and how complex the evaluation of such queries is. In this section, we study the expressive power of SPARQL. The main goal is to show that SPARQL is equivalent, from an expressive-power point of view, to Relational Algebra.

In order to determine the expressive power of a query language $\mathcal{L}$, one usually chooses a well-studied query language $\mathcal{L}'$, and then compares the expressiveness of $\mathcal{L}$ and $\mathcal{L}'$. In particular, one says that two query languages have the same expressive power if they express exactly the same set of queries. In this section, we present an overview of the results in [7], that show that the query language SPARQL SELECT has the same expressiveness as non-recursive Datalog with negation (nr-Datalog$^\neg$) and Relational Algebra.

We start with an overview of Datalog (for further details see [1,33]). A *term* is either a *variable* or a *constant*. An *atom* is either a *predicate formula* $p(x_1, ..., x_n)$, where p is a predicate name and each x_i is a term, or an *equality formula* $t_1 = t_2$, where t_1 and t_2 are terms. A *literal* is either an atom (a *positive literal*), or the negation of an atom (a *negative literal*). A *fact* is a predicate formula containing only constants. A *substitution* θ for variables $x_1, \ldots, x_k$ is a set of assignments $\{x_1 \to t_1, \ldots, x_k \to t_k\}$ where each t_i is a term. Given a literal L, we denote by $\theta(L)$ the literal that results by replacing in L each variable x_i by the term t_i.

A Datalog *rule* is an expression $H \leftarrow L_1, \ldots, L_n$, where H is a predicate formula containing only variables and each L_i is a literal. H is called the *head* of the rule, and the sequence $L_1, \ldots, L_n$ is called its *body*. A *Datalog program* Π is a finite set of Datalog rules. A predicate is *extensional* in Π if it does not occur in the head of any rule of Π, otherwise it is called *intensional*. A Datalog program is *non-recursive* if there is some ordering $r_1, \ldots, r_m$ of its rules so that, the predicate name in the head of r_i does not occur in the body of a rule r_j for every $j \leq i$. We further impose the following *safety* condition to rules: every variable occurring in a rule r must occur in at least one (positive) predicate formula in the body of r. In what follows, we only consider non-recursive and safe programs. Moreover, we may assume that all heads of rules in a program have distinct variables, since repeated variables can always be replaced by adding equalities. For example, the rule $p(X, X) \leftarrow t(X)$ can be replaced by $p(X, Y) \leftarrow t(X), t(Y), X = Y$.

Let D be a set of facts over the extensional predicates of a Datalog program Π. We define the *meaning* of Π given D, denoted by $\text{facts}^*(\Pi, D)$, as the set of facts that results from the following process. Fix an order $r_1, \ldots, r_m$ of the rules that satisfies the aforementioned non-recursive property. The set $\text{facts}^*(\Pi, D)$ is obtained evaluating the rules by following that order. Formally, we denote by $\text{facts}^i(\Pi, D)$ the total set of facts obtained after evaluating rule r_i. Initially, $\text{facts}^0(\Pi, D) = D$. In order to compute $\text{facts}^{i+1}(\Pi, D)$, assume that rule r_{i+1} is $H \leftarrow L_1, \ldots L_n$. Then $\text{facts}^{i+1}(\Pi, D)$ is obtained by adding to $\text{facts}^i(\Pi, D)$ all the facts of the form $\theta(H)$, where θ is a substitution such that $\theta(L_1), \ldots, \theta(L_n)$ hold in $\text{facts}^i(\Pi, D)$. The process stops when all rules have been considered.

A *Datalog query* Q is a pair (Π, L) where Π is a Datalog program and L is a predicate formula (the *goal* of the program). The *answer* to a Datalog query $Q = (\Pi, L)$ over a database D, denoted by $\underline{\text{answer}}(Q, D)$, is the set of all substitutions θ for the variables occurring in L, such that $\theta(L) \in \text{facts}^*(\Pi, D)$.

5.1 From SPARQL to nr-Datalog$^\neg$

In this section, we show that nr-Datalog$^\neg$ is at least as expressive as SPARQL SELECT, that is, we show that every SPARQL SELECT query can be expressed as an nr-Datalog$^\neg$ program. More specifically, we first define a one-to-one transformation $\mathcal{T}_1$ that assigns to every RDF graph G a set of Datalog facts $\mathcal{T}_1(G)$. We then define a one-to-one transformation $\mathcal{T}_2$ that assigns to every SPARQL SELECT query Q, a Datalog query $\mathcal{T}_2(Q)$, and show that for every SPARQL SELECT query Q and RDF graph G, the evaluation of Q over G corresponds to the evaluation of the Datalog query $\mathcal{T}_2(Q)$ over the set of facts $\mathcal{T}_1(G)$.

The transformation $\mathcal{T}_1$ from RDF graphs into Datalog facts essentially transform triples into facts, but taking special care of encoding unbounded values as nulls. Formally, given an RDF graph G, the transformation $\mathcal{T}_1(G)$ works as follows: every element a occurring in G is encoded by a fact $\textit{term}(a)$; each triple (s, p, o) is encoded by a fact $\textit{triple}(s, p, o)$; additionally, we include a special fact $\mathbf{N}(\textit{null})$, where *null* is a constant value used to represent unbounded variables.

We now have to show how graph patterns are transformed into Datalog rules. We show here some examples of this transformation to highlight the intuition of the process. We refer the reader to [44,7] for the details on the general transformation. Consider first the graph pattern $P_1 = ((?X, a, 1)\ \text{OPT}\ (?X, b, ?Z))$. Then the transformation $\mathcal{T}_2$ generates the following Datalog program with goal predicate p to express P_1:

$$p(?X, ?Z) \leftarrow \textit{triple}(?X, a, 1), \textit{triple}(?X, b, ?Z) \tag{6}$$
$$p(?X, ?Z) \leftarrow \textit{triple}(?X, a, 1), \mathbf{N}(?Z), \neg q(?X) \tag{7}$$
$$q(?X) \leftarrow \textit{triple}(?X, b, ?V) \tag{8}$$

The first rule is encoding the *join* operation between sets of mappings, while the second and third rules are encoding the *difference*. The left outer-join, which defines the semantics of the OPT operator, is then obtained by considering rules (6), (7) and (8), that is, considering the union between the results of the join and the difference. Notice that predicate $\mathbf{N}$ is used in the second rule to encode unbounded variables.

Second, consider SPARQL SELECT query $(\{?Z\}, P_1)$, where P_1 is the pattern defined above. To express the SELECT operator, one only needs to perform a *projection* in Datalog, that is, one can express query $(\{?Z\}, P_1)$ by using rules (6), (7), (8) and the following projection rule:

$$r(?Z) \leftarrow p(?X, ?Z).$$

Notice that in this case r is the new goal predicate.

Finally, consider SPARQL pattern:

$$P_2 = \Big((?X, a, 1) \text{ AND } \Big((?X, b, 1) \text{ UNION } (?Y, c, 1)\Big)\Big).$$

The main difficulty in translating P_2 into an nr-Datalog$^\neg$ program is the encoding of the notion of *compatible mapping*. To see why this is the case, first notice that one can easily express pattern $P_2' = ((?X, b, 1) \text{ UNION } (?Y, c, 1))$ as an nr-Datalog$^\neg$ program:

$$\begin{aligned} p'(?X, ?Y) &\leftarrow \mathit{triple}(?X, b, 1), \mathbf{N}(?Y), \\ p'(?X, ?Y) &\leftarrow \mathit{triple}(?Y, c, 1), \mathbf{N}(?X). \end{aligned}$$

But if we now want to translate pattern $P_2 = ((?X, a, 1) \text{ AND } P_2')$, one cannot directly use the previous two rules together with a rule like the following:

$$p(?X, ?Y) \leftarrow \mathit{triple}(?X, a, 1), p'(?X, ?Y),$$

as this rule does not take into consideration the fact that the occurrence of $?X$ in p' could be instantiated with value *null*. In fact, if this is the case, then the rule does not generate any facts as either there is no value $d \in U$ such that $\mathit{triple}(d, a, 1)$ holds, or there is such a value d but then d is different from *null*. Notice that this failure is due to the fact that the previous rule does not correctly encode the notion of compatible mapping. To solve this problem, one needs to replace the previous rule by:

$$p(?X, ?Y) \leftarrow \mathit{triple}(?X, a, 1), p'(?U, ?Y), \mathit{compatible}(?X, ?U),$$

where $\mathit{compatible}(\cdot, \cdot)$ is defined as:

$$\begin{aligned} \mathit{compatible}(?X, ?Y) &\leftarrow \mathit{term}(?X), \mathit{term}(?Y), ?X = ?Y \\ \mathit{compatible}(?X, ?Y) &\leftarrow \mathit{term}(?X), \mathbf{N}(?Y) \\ \mathit{compatible}(?X, ?Y) &\leftarrow \mathbf{N}(?X), \mathit{term}(?Y) \\ \mathit{compatible}(?X, ?Y) &\leftarrow \mathbf{N}(?X), \mathbf{N}(?Y) \end{aligned}$$

To conclude this section, it only remains to show how SPARQL mappings are represented as Datalog substitutions. Notice that a mapping μ is a partial function. To represent the fact that a mapping is not defined for some variables, we use the special value *null*. Given a mapping μ and a set of variables W such that $\mathrm{dom}(\mu) \subseteq W$, we define $\theta_{(\mu,W)}$ as a substitution for variables in W such that (1) $\theta_{(\mu,W)}(?X) = \mu(?X)$ for every variable $?X \in \mathrm{dom}(\mu)$, and (2) $\theta_{(\mu,W)}(?X) = \mathit{null}$ for every variable $?X$ such that $?X \in W$ and $?X \notin \mathrm{dom}(\mu)$.

With the above transformations, we can show that nr-Datalog$^{\neg}$ is at least as expressive as the language SPARQL SELECT. More precisely, let G be an RDF graph and $Q = (W, P)$ a SPARQL SELECT query, with W a set of variables and P a SPARQL graph pattern. Then a mapping μ is in $[\![Q]\!]_G$ if and only if the substitution $\theta_{(\mu,W)}$ is in $\underline{\mathrm{answer}}(\mathcal{T}_1(Q), \mathcal{T}_2(G))$. Thus, we have that:

Theorem 10 ([44,7]). *nr-Datalog$^{\neg}$ is at least as expressive as the language* SPARQL SELECT*.*

5.2 From Datalog to SPARQL

In this section, we show that SPARQL is at least as expressive as nr-Datalog$^{\neg}$, that is, we provide transformations from Datalog facts into RDF graphs, Datalog substitutions into SPARQL mappings, and nr-Datalog$^{\neg}$ programs into SPARQL graph patterns. But before presenting these transformations, we give a technical result that is used to encode negated literals of Datalog rules. Let MINUS be a binary operator defined as follows. Given SPARQL graph patterns P_1, P_2 and an RDF graph G:

$$[\![(P_1 \text{ MINUS } P_2)]\!]_G = [\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G,$$

where $\smallsetminus$ denotes the difference between sets of mappings defined in Section 3. Then the following proposition shows that the MINUS operator can be expressed in SPARQL:

Proposition 6. *Let P_1 and P_2 be graph patterns. Then pattern* $(P_1 \text{ MINUS } P_2)$ *is equivalent to:*

$$\Bigg(\Bigg(P_1 \text{ OPT } (P_2 \text{ AND } (?X_1, ?X_2, ?X_3))\Bigg) \text{ FILTER } \neg\,\mathrm{bound}(?X_1)\Bigg), \qquad (9)$$

where $?X_1, ?X_2, ?X_3$ are fresh variables mentioned neither in P_1 nor in P_2.

Thus, from now on we use SPARQL patterns including the operator MINUS, as they can be translated into usual SPARQL patterns.

We now describe the transformations used to show that nr-Datalog$^{\neg}$ is contained in SPARQL. Given a fact $f = p(c_1, ..., c_n)$, let $\mathrm{desc}(f)$ be the set of triples $\{(b,\ \text{predicate},\ p), (b, 1, c_1), \ldots, (b, n, c_n)\}$, where b is a *fresh* value in U. Moreover, given a set of facts D, define a one-to-one transformation $\mathcal{T}_1'$ as $\mathcal{T}_1'(D) = \{\mathrm{desc}(f) \mid f \in D\}$.

Transformation $\mathcal{T}_1'$ allows one to represent a set of facts as an RDF graph. Thus, to show that SPARQL SELECT is at least as expressive as nr-Datalog$^{\neg}$, it remains to provide a one-to-one mapping $\mathcal{T}_2'$ that transforms nr-Datalog$^{\neg}$ programs into SPARQL SELECT queries. As we did for the other direction, we show the intuition of the transformation with an example, and refer the reader to [7] for a detailed description of this transformation. Let $\mathit{\Pi}$ be an nr-Datalog$^\neg$ program, and L a predicate formula $p(x_1, \ldots, x_n)$. For the sake of readability, we assume that all the variables in $\mathit{\Pi}$ are in V (that is, they can be used as variables in SPARQL graph patterns). We define $\text{gp}(\mathit{\Pi}, L)$ as a function which returns a graph pattern that encodes the program $(\mathit{\Pi}, L)$. The function $\text{gp}(\mathit{\Pi}, L)$ works as follows:

(a) If predicate p is extensional in $\mathit{\Pi}$, then $\text{gp}(\mathit{\Pi}, L)$ returns the graph pattern $((?Y, \text{predicate}, p) \text{ AND } (?Y, 1, x_1) \text{ AND } \cdots \text{ AND } (?Y, 1, x_n))$, where $?Y$ is a fresh variable.
(b) If predicate p is intensional in $\mathit{\Pi}$, then for each rule $L \leftarrow L_1, \cdots, L_s, \neg K_1, \cdots, \neg K_t, L_1^{eq}, \cdots, L_u^{eq}$ in $\mathit{\Pi}$ having p in its head, where each L_i is a positive literal and each L_j^{eq} is a literal of the form $t_1 = t_2$ or $\neg(t_1 = t_2)$, the following SPARQL pattern is generated:

$$\Bigg[\bigg(\bigg(\cdots\bigg(\bigg(\text{gp}(\mathit{\Pi}, L_1) \text{ AND } \cdots \text{ AND } \text{gp}(\mathit{\Pi}, L_s)\bigg) \text{ MINUS } \text{gp}(\mathit{\Pi}, K_1)\bigg)\cdots\bigg) \text{ MINUS } \text{gp}(\mathit{\Pi}, K_t)\bigg) \text{ FILTER } \bigg(L_1^{eq} \wedge \cdots \wedge L_u^{eq}\bigg)\Bigg].$$

Assume that there are k rules in $\mathit{\Pi}$ having p in their heads, and that P_1, …, P_k are the SPARQL patterns generated from these rules as above. Then $\text{gp}(\mathit{\Pi}, L)$ is defined as $(P_1 \text{ UNION } \cdots \text{ UNION } P_k)$.

Function $\text{gp}(\cdot, \cdot)$ is used to define transformation $\mathcal{T}_2'$. More precisely, if the set of variables mentioned in L is W, then $\mathcal{T}_2'((\mathit{\Pi}, L))$ is the SPARQL SELECT query $(W, \text{gp}(\mathit{\Pi}, L))$.

Example 5. Consider the following Datalog program $\mathit{\Pi}$:

$$\begin{aligned} p(?X, ?Y) &\leftarrow r(?X, ?Y, ?Z), \neg s(?X, ?X) \\ p(?X, ?Y) &\leftarrow t(?X, ?Y) \end{aligned}$$

In order to translate this program into a SPARQL SELECT query, the first rule is transformed into the pattern:

$$P_1 = \Bigg[\bigg((?U, \text{predicate}, r) \text{ AND } (?U, 1, ?X) \text{ AND } (?U, 2, ?Y) \text{ AND } (?U, 3, ?Z)\bigg) \text{ MINUS } \bigg((?V, \text{predicate}, s) \text{ AND } (?V, 1, ?X) \text{ AND } (?V, 2, ?X)\bigg)\Bigg],$$

and the second rule is transformed into the pattern:

$$P_2 = \bigg((?W, \text{predicate}, t) \text{ AND } (?W, 1, ?X) \text{ AND } (?W, 2, ?Y)\bigg).$$

Thus, we have that $\text{gp}(\Pi, p(?X, ?Y))$ is the pattern $(P_1 \text{ UNION } P_2)$, from which we conclude that $\mathcal{T}_2'((\Pi, p(?X, ?Y)))$ is the SPARQL SELECT query $(\{?X, ?Y\}, (P_1 \text{ UNION } P_2))$. $\square$

To conclude this section, it only remains to show how Datalog substitutions are represented as SPARQL mappings. Given a substitution θ over a set W of variables, define μ_θ as a mapping such that: (1) $?X \in \text{dom}(\mu_\theta)$ if and only if $?X \to t$ is in θ and $t \neq \mathit{null}$, and (2) for every $?X \in \text{dom}(\mu_\theta)$, mapping μ_θ assigns to $?X$ the value assigned by θ to this variable. This transformation together with $\mathcal{T}_1'$ and $\mathcal{T}_2'$ can be used to show that the language SPARQL SELECT is at least as expressive as nr-Datalog$^\neg$. More precisely, given a set D of Datalog facts and an nr-Datalog$^\neg$ query $Q = (\Pi, L)$, we have that a substitution θ is in $\underline{\text{answer}}(Q, D)$ if and only if the mapping μ_θ is in $[\![\mathcal{T}_2'(Q)]\!]_{\mathcal{T}_1'(D)}$. Thus, we have that:

Theorem 11 ([7]). *The language* SPARQL SELECT *is at least as expressive as nr-Datalog*$^\neg$*.*

From Theorems 10 and 11, and using the well-known fact that Relational Algebra has the same expressive power as nr-Datalog$^\neg$ [1], we obtain that SPARQL SELECT and Relational Algebra have the same expressive power.

Corollary 3 ([7]). *The language* SPARQL SELECT *has the same expressive power as Relational Algebra.*

6 A Query Language for RDFS Data

The RDF specification includes a set of reserved keywords with its own semantics, the RDFS vocabulary. This vocabulary is designed to describe special relationships between resources like typing and inheritance of classes and properties [11]. As with any data structure designed to model information, a natural question that arises is what the desiderata are for an RDFS query language. Among the multiple design issues to be considered, it has been largely recognized that navigational capabilities are of fundamental importance for data models with explicit tree or graph structure (like XML and RDF [6,12]).

SPARQL has been designed much in the spirit of classical relational languages such as SQL. In particular, it has been noted that, although RDF is a directed labeled graph data format, SPARQL only provides limited navigational functionalities. This is more notorious when one considers the RDFS vocabulary (which current SPARQL specification does not cover [45]), where testing conditions like being a subclass of or a subproperty of naturally requires navigating the RDF data. A good illustration of this is shown by the following query, which cannot be expressed in SPARQL without some navigational capabilities. Consider the RDF graph shown in Fig. 3. This graph stores information about cities, transportation services between cities, and further relationships among those transportation services (in the form of RDFS annotations). For instance, in the graph we have that a "Seafrance" service is a subproperty of a "ferry" service, which in turn is a subproperty of a general "transport" service. Assume that we

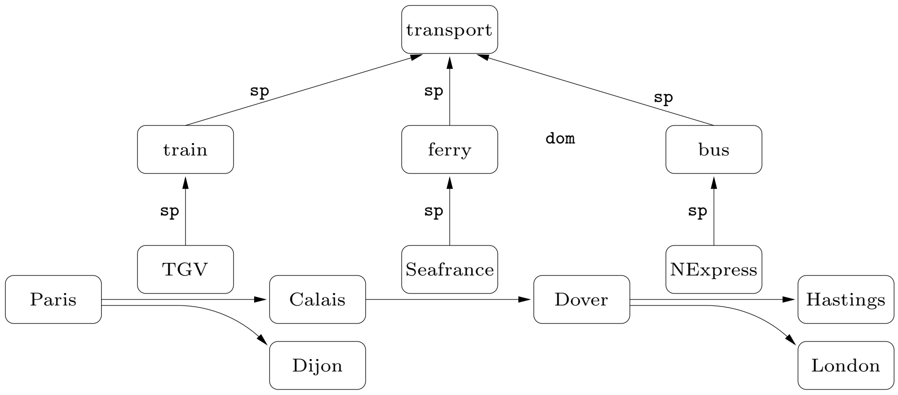


Fig. 3. An RDF graph storing information about transportation services between cities

want to test whether a pair of cities A and B are connected by a sequence of transportation services, but without knowing in advance what services provide those connections. We can answer such a query by testing whether there is a path connecting A and B in the graph, such that every edge in that path is connected with "transport" by following a sequence of subproperty relationships. For instance, for "Paris" and "Calais" the condition holds, since "Paris" is connected with "Calais" by an edge with label "TGV", and "TGV" is a subproperty of "train", which in turn is a subproperty of "transport". Notice that the condition also holds for "Paris" and "Dover".

In this section, we present a language for navigating RDF data grounded on paths expressed with regular expressions, which was proposed in [43]. This language takes advantage of the special features of RDF, and besides regular expressions, it borrows the notion of *branching* from XPath [17], to obtain what is called *nested regular expressions*. We also show how these navigational capabilities can be incorporated into SPARQL, which gives rise to the query language nSPARQL [43].

Furthermore, in this section we consider two fundamental questions about these new navigational capabilities and the language nSPARQL. First, we deal with the problem of whether these new navigational capabilities can be implemented efficiently. In this section, we present the evaluation algorithm for nested regular expressions that was proposed in [43], and which works in time $O(|G| \cdot |E|)$ for an RDF graph G and a nested regular expression E. Second, we consider the issue of whether nSPARQL is a *good* query language from an expressiveness point of view. In this section, we provide evidence that the capabilities of nSPARQL can be used to pose many interesting and natural queries over RDF data. For the sake of presentation, in this section we consider RDF graphs constructed by using only elements from U, that is, we do not consider blank nodes.

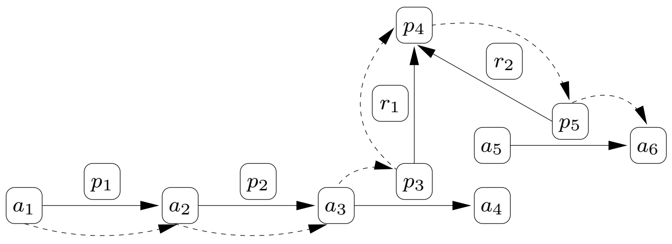
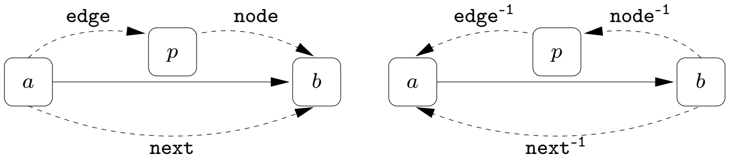


Fig. 4. Nodes a_1 and a_6 are connected by a path that follows the sequence of navigational axes `next/next/edge/next/next`$^{-1}$`/node`

6.1 Nested Regular Expressions for RDF Data

As usual for graph query languages [36,14,6], the language presented in this section uses regular expressions to define paths on graph structures, but taking advantage of the special features of RDF graphs.

The navigation of a graph is usually done by using an operator *next*, which allows one to move from one node to an adjacent one. In our setting, we have RDF "graphs", which are sets of triples, not classical graphs. In particular, instead of classical edges (pair of nodes), we have directed triples of nodes (*hyperedges*). Hence, a language for navigating RDF graphs should be able to deal with this type of objects. In this section, we present the notion of *nested regular expression* to navigate through an RDF graph, which was introduced in [43]. This notion takes into account the special features of the RDF data model. In particular, nested regular expressions use three different *navigation axes* `next`, `edge` and `node`, and their inverses `next`$^{-1}$, `edge`$^{-1}$ and `node`$^{-1}$, to move through an RDF triple. These axes are shown in the following figure:



A navigation axis allows one to move one step forward (or backward) in an RDF graph. Thus, a sequence of these axes defines a path in an RDF graph. For instance, in the graph of Fig. 4, the sequence of axes:

`next/next/edge/next/next`$^{-1}$`/node`

defines a path between nodes a_1 and a_6 (the path is shown with dashed lines in the figure). Moreover, one can use classical regular expressions over these axes to define a set of paths that can be used in a query. The language proposed in [43] considers an additional axis `self` that is used not to actually navigate, but instead to test the label of a specific node in a path. The language also allows *nested expressions* that can be used to test for the existence of certain paths starting at any axis. The following grammar defines the syntax of nested regular expressions:

$$\begin{aligned} exp \;:=\; & \text{axis} \;\mid\; \text{axis}::a \; (a \in U) \;\mid\; \text{axis}::[exp] \;\mid \\ & exp/exp \;\mid\; exp|exp \;\mid\; exp^* \end{aligned} \qquad (10)$$

where axis $\in \{\texttt{self}, \texttt{next}, \texttt{next}^{-1}, \texttt{edge}, \texttt{edge}^{-1}, \texttt{node}, \texttt{node}^{-1}\}$. Before introducing the formal semantics of nested regular expressions, we give some intuition about how these expressions are evaluated in an RDF graph. The most natural navigation axis is $\texttt{next}::a$, with a an arbitrary element from U. Given an RDF graph G, the expression $\texttt{next}::a$ is interpreted as the *a-neighbor* relation in G, that is, the pairs of nodes (x, y) such that $(x, a, y) \in G$. Given that in the RDF data model, a node can also be the label of an edge, the language allows one to navigate from a node to one of its leaving edges by using the `edge` axis. More formally, the interpretation of $\texttt{edge}::a$ is the pairs of nodes (x, y) such that $(x, y, a) \in G$. The nesting construction $[exp]$ is used to check for the existence of a path defined by expression *exp*. For instance, when evaluating nested expression $\texttt{next}::[exp]$ in a graph G, we retrieve the pairs of nodes (x, y) such that there exists z with $(x, z, y) \in G$, and such that there is a path in G that follows expression *exp* starting in z.

The evaluation of a nested regular expression *exp* in a graph G is formally defined as a binary relation $[\![exp]\!]_G$, denoting the pairs of nodes (x, y) such that y is reachable from x in G by following a path that conforms to *exp* [43]. The formal semantics of the language is shown in Tab. 2. In this table, G is an RDF graph, $a \in U$, voc(G) is the set of all the elements from U that are mentioned in G, and *exp*, exp_1, exp_2 are nested regular expressions.

Example 6. Let G be the graph in Fig. 3, and consider expression

$$exp_1 = \texttt{next}::[\texttt{next}::\texttt{sp}/\texttt{self}::\text{train}].$$

The expression $\texttt{next}::\texttt{sp}/\texttt{self}::\text{train}$ defines the pairs of nodes (z, w) such that from z one can reach w by following an edge labeled `sp`, and furthermore the label of w is train (expression $\texttt{self}::\text{train}$ is used to perform this test). Thus, the nested expression $[\texttt{next}::\texttt{sp}/\texttt{self}::\text{train}]$ performs an existential test; it is satisfied by the nodes in G from which there exists a path that follows an edge labeled `sp` and reaches a node labeled train. TGV is the only such node in G and, thus, we have that $[\![exp_1]\!]_G = \{(\text{Paris, Calais}), (\text{Paris, Dijon})\}$. □

6.2 An Efficient Algorithm for Evaluating Nested Regular Expressions

In [43], it was introduced the language nSPARQL that combines the operators of SPARQL with the navigational capabilities of nested regular expressions. As pointed out in that paper, an essential requirement to use nSPARQL in large applications is that nested regular expressions could be evaluated efficiently.

Table 2. Formal semantics of nested regular expressions

$$
\begin{aligned}
[\![\mathtt{self}]\!]_G &= \{(x,x) \mid x \in \mathrm{voc}(G)\}\\
[\![\mathtt{self}::a]\!]_G &= \{(a,a)\}\\
[\![\mathtt{next}]\!]_G &= \{(x,y) \mid \text{there exists } z \text{ s.t. } (x,z,y) \in G\}\\
[\![\mathtt{next}::a]\!]_G &= \{(x,y) \mid (x,a,y) \in G\}\\
[\![\mathtt{edge}]\!]_G &= \{(x,y) \mid \text{there exists } z \text{ s.t. } (x,y,z) \in G\}\\
[\![\mathtt{edge}::a]\!]_G &= \{(x,y) \mid (x,y,a) \in G\}\\
[\![\mathtt{node}]\!]_G &= \{(x,y) \mid \text{there exists } z \text{ s.t. } (z,x,y) \in G\}\\
[\![\mathtt{node}::a]\!]_G &= \{(x,y) \mid (a,x,y) \in G\}\\
[\![\mathrm{axis}^{\text{-}1}]\!]_G &= \{(x,y) \mid (y,x) \in [\![\mathrm{axis}]\!]_G\} \quad \text{with axis} \in \{\mathtt{next},\mathtt{node},\mathtt{edge}\}\\
[\![\mathrm{axis}^{\text{-}1}::a]\!]_G &= \{(x,y) \mid (y,x) \in [\![\mathrm{axis}::a]\!]_G\} \quad \text{with axis} \in \{\mathtt{next},\mathtt{node},\mathtt{edge}\}\\
[\![\mathit{exp}_1/\mathit{exp}_2]\!]_G &= \{(x,y) \mid \text{there exists } z \text{ s.t. } (x,z) \in [\![\mathit{exp}_1]\!]_G \text{ and } (z,y) \in [\![\mathit{exp}_2]\!]_G\}\\
[\![\mathit{exp}_1|\mathit{exp}_2]\!]_G &= [\![\mathit{exp}_1]\!]_G \cup [\![\mathit{exp}_2]\!]_G\\
[\![\mathit{exp}^*]\!]_G &= [\![\mathtt{self}]\!]_G \cup [\![\mathit{exp}]\!]_G \cup [\![\mathit{exp}/\mathit{exp}]\!]_G \cup [\![\mathit{exp}/\mathit{exp}/\mathit{exp}]\!]_G \cup \cdots\\
[\![\mathtt{self}::[\mathit{exp}]]\!]_G &= \{(x,x) \mid x \in \mathrm{voc}(G) \text{ and there exists } z \text{ s.t. } (x,z) \in [\![\mathit{exp}]\!]_G\}\\
[\![\mathtt{next}::[\mathit{exp}]]\!]_G &= \{(x,y) \mid \text{there exist } z,w \text{ s.t. } (x,z,y) \in G \text{ and } (z,w) \in [\![\mathit{exp}]\!]_G\}\\
[\![\mathtt{edge}::[\mathit{exp}]]\!]_G &= \{(x,y) \mid \text{there exist } z,w \text{ s.t. } (x,y,z) \in G \text{ and } (z,w) \in [\![\mathit{exp}]\!]_G\}\\
[\![\mathtt{node}::[\mathit{exp}]]\!]_G &= \{(x,y) \mid \text{there exist } z,w \text{ s.t. } (z,x,y) \in G \text{ and } (z,w) \in [\![\mathit{exp}]\!]_G\}\\
[\![\mathrm{axis}^{\text{-}1}::[\mathit{exp}]]\!]_G &= \{(x,y) \mid (y,x) \in [\![\mathrm{axis}::[\mathit{exp}]]\!]_G\} \quad \text{with axis} \in \{\mathtt{next},\mathtt{node},\mathtt{edge}\}
\end{aligned}
$$

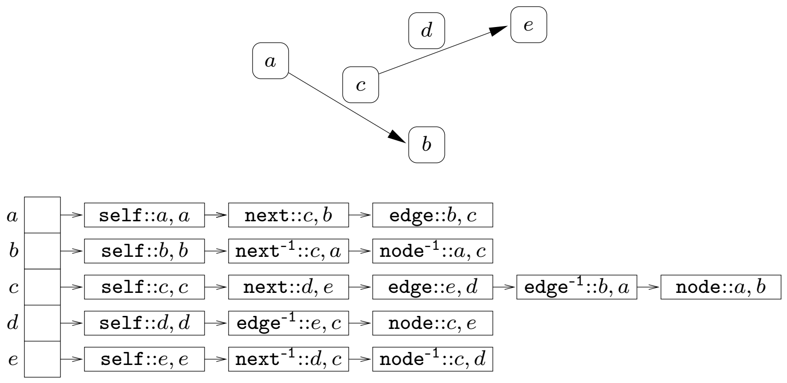
In this section, we present an efficient algorithm for this task, which works in time proportional to the size of the input graph times the size of the expression being evaluated. As is customary when studying the complexity of the evaluation problem for a query language [51], we consider its associated decision problem. For nested regular expressions, this problem is defined as:

PROBLEM	: Evaluation problem for nested regular expressions.
INPUT	: An RDF graph G, a nested regular expression *exp*, and a pair (a,b).
QUESTION	: Is $(a,b) \in [\![\mathit{exp}]\!]_G$?

It is important to note that the evaluation problem that we study considers the pair of nodes (a,b) as part of the input. That is, similar to the complexity study presented in Section 4, we study the complexity by measuring how difficult it is to verify whether a given pair of nodes is in the evaluation of a nested regular expression over an RDF graph.

Following the terminology introduced in [43], we assume that an RDF graph G is stored as an adjacency list that makes explicit the navigation axes (and their inverses). Thus, every $u \in \mathrm{voc}(G)$ is associated with a list of pairs $\alpha(u)$, where every pair contains a navigation axis and the destination node. For instance, if (s,p,o) is a triple in G, then $(\mathtt{next}::p, o) \in \alpha(s)$ and $(\mathtt{edge}^{\text{-}1}::o, s) \in \alpha(p)$. Moreover, we assume that $(\mathtt{self}::u, u) \in \alpha(u)$ for every $u \in \mathrm{voc}(G)$. Notice that if the number of triples in G is N, then the adjacency list representation uses space $O(N)$. Thus, when measuring the size of G, we use $|G|$ to denote the size of its adjacency list representation. We further assume that given an element $u \in \text{voc}(G)$, we can access its associated list $\alpha(u)$ in time $O(1)$. This is a standard assumption for graph data-structures in a RAM model [19].

Example 7. The following figure shows an example of an adjacency-list representation of an RDF graph.



a
b
c
d
e
self::a, a → next::c, b → edge::b, c
self::b, b → next^{-1}::c, a → node^{-1}::a, c
self::c, c → next::d, e → edge::e, d → edge^{-1}::b, a → node::a, b
self::d, d → edge^{-1}::e, c → node::c, e
self::e, e → next^{-1}::d, c → node^{-1}::c, d

□

The algorithm in [43] for the evaluation of nested regular expressions was inspired by some of the algorithms for the evaluation of temporal logics [18] and propositional dynamic logic [2,24]. To present this algorithm, we need to introduce some terminology. An expression *exp*$'$ is a *nested subexpression* of an expression *exp* if axis::[*exp*$'$] occurs in *exp*, with axis $\in$ {`self`, `next`, `next`$^{-1}$, `edge`, `edge`$^{-1}$, `node`, `node`$^{-1}$}. Given an RDF graph G and a nested regular expression *exp*, the algorithm proceeds by recursively considering the nested subexpressions of *exp*, labeling every node u of G with a set label(u) of nested expressions. Initially, label(u) is the empty set. Then at the end of the execution of the algorithm, it holds that $exp \in \text{label}(u)$ if and only if there exists z such that $(u, z) \in [\![exp]\!]_G$. Before giving any technical details, let us show the general idea of this process with an example. Figure 5 exemplifies the process for a graph G and the nested expression:

$$\beta = \texttt{next}::a/(\texttt{next}::[\texttt{next}::b/\texttt{self}::c])^*/(\texttt{edge}::[\texttt{next}::d] \mid \texttt{next}::a)^+. \quad (11)$$

The process first considers the nested subexpressions $\gamma = \texttt{next}::b/\texttt{self}::c$ and $\lambda = \texttt{next}::d$, and marks the nodes in G according to which ones of these subexpressions they satisfy. Thus, after this stage we have that $\gamma \in \text{label}(r_3)$ since $(r_3, c) \in [\![\gamma]\!]_G$, and $\lambda \in \text{label}(r_6)$ since $(r_6, r_7) \in [\![\lambda]\!]_G$ (see Fig. 5). Using this information, the nodes are marked according to whether they satisfy β, but considering the previously computed labels (γ and λ) and the expression $\beta' = \texttt{next}::a/(\texttt{next}::\gamma)^*/(\texttt{edge}::\lambda \mid \texttt{next}::a)^+$. In the example of Fig. 5, we have that $(r_1, r_5) \in [\![\beta]\!]_G$ and, thus, $\beta \in \text{label}(r_1)$.

We now explain how to efficiently carry out the labeling process by using some tools from automata theory (here we assume some familiarity with this theory). A key idea in the algorithm presented in [43] is to associate to each nested regular expression a nondeterministic finite automaton with ε-transitions

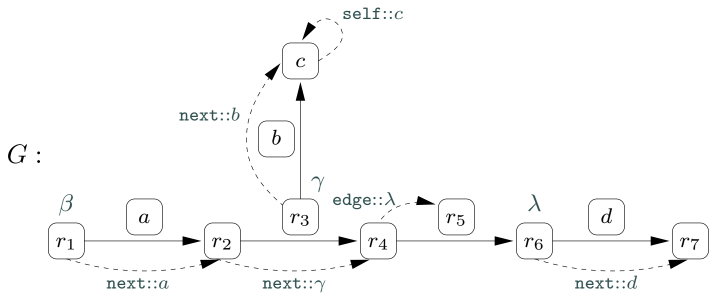


Fig. 5. Example of the labeling process of the RDF graph G according to expression $\beta = \texttt{next}::a/(\texttt{next}::[\texttt{next}::b/\texttt{self}::c])^*/(\texttt{edge}::[\texttt{next}::d] \mid \texttt{next}::a)^+$. First, node r_3 is marked with label $\gamma = \texttt{next}::b/\texttt{self}::c$ (since $(r_3, c) \in [\![\gamma]\!]_G$), and node r_6 with label $\lambda = \texttt{next}::d$ (since $(r_6, r_7) \in [\![\lambda]\!]_G$). Finally, node r_1 is labeled with β (since $(r_1, r_5) \in [\![\beta]\!]_G$). This last label is obtained by considering the expression $\beta' = \texttt{next}::a/(\texttt{next}::\gamma)^*/(\texttt{edge}::\lambda \mid \texttt{next}::a)^+$.

(ε-NFA). Given a nested regular expression *exp*, the set of *depth-0 terms* of *exp*, denoted by $\mathbf{D}_0(\mathit{exp})$, is recursively defined as follows:

$$\mathbf{D}_0(\mathit{exp}) = \{\mathit{exp}\} \text{ if } \mathit{exp} \text{ is either axis, or axis::}a\text{, or axis::}[\,\mathit{exp}'\,],$$
$$\mathbf{D}_0(\mathit{exp}_1/\mathit{exp}_2) = \mathbf{D}_0(\mathit{exp}_1|\mathit{exp}_2) = \mathbf{D}_0(\mathit{exp}_1) \cup \mathbf{D}_0(\mathit{exp}_2),$$
$$\mathbf{D}_0(\mathit{exp}^*) = \mathbf{D}_0(\mathit{exp}),$$

where axis $\in \{\texttt{self}, \texttt{next}, \texttt{next}^{-1}, \texttt{edge}, \texttt{edge}^{-1}, \texttt{node}, \texttt{node}^{-1}\}$. For instance, for the nested expression β in (11), we have that:

$$\mathbf{D}_0(\beta) = \{\,\texttt{next}::a,\ \texttt{next}::[\texttt{next}::b/\texttt{self}::c],\ \texttt{edge}::[\texttt{next}::d]\,\}.$$

Notice that a nested regular expression *exp* can be viewed as a classical regular expression over the alphabet $\mathbf{D}_0(\mathit{exp})$. We denote by $\mathcal{A}_{\mathit{exp}}$ the ε-NFA that accepts the language generated by the regular expression *exp* over the alphabet $\mathbf{D}_0(\mathit{exp})$. For example, Fig. 6 shows an ε-NFA $\mathcal{A}_\beta$ that accepts the language generated by expression β in (11) over the alphabet $\mathbf{D}_0(\beta)$. As for the case of RDF graphs, ε-NFAs are stored using an adjacency-list representation.

An essential ingredient in the algorithm presented in [43] is the use of the product automaton $G \times \mathcal{A}_{\mathit{exp}}$, which is constructed as follows. Assume that we have the graph G labeled with respect to the nested subexpressions of *exp*, that is, for every node u of G and nested subexpression exp' of *exp*, we have that $\mathit{exp}' \in \text{label}(u)$ if and only if there exists a node v such that $(u, v) \in [\![\mathit{exp}']\!]_G$. Let Q be the set of states of $\mathcal{A}_{\mathit{exp}}$, and $\delta : Q \times (\mathbf{D}_0(\mathit{exp}) \cup \{\varepsilon\}) \to 2^Q$ the transition function of $\mathcal{A}_{\mathit{exp}}$. Then the set of states of $G \times \mathcal{A}_{\mathit{exp}}$ is $\text{voc}(G) \times Q$, and its transition function $\delta' : (\text{voc}(G) \times Q) \times (\mathbf{D}_0(\mathit{exp}) \cup \{\varepsilon\}) \to 2^{\text{voc}(G) \times Q}$ is defined as follows. For every $(u, p) \in \text{voc}(G) \times Q$ and $s \in \mathbf{D}_0(\mathit{exp})$, we have that $(v, q) \in \delta'((u, p), s)$ if and only if $q \in \delta(p, s)$ and one of the following cases hold:

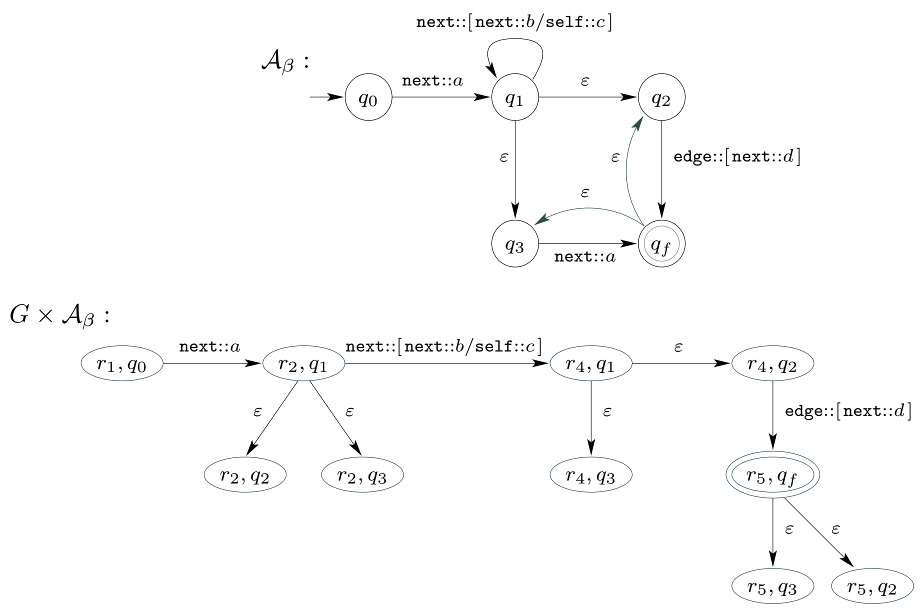


Fig. 6. Automaton $\mathcal{A}_\beta$ for the nested regular expression β in ([11]), and product automaton $G \times \mathcal{A}_\beta$

- $s =$ axis and there exists a such that $(\text{axis}::a, v) \in \alpha(u)$,
- $s = \text{axis}::a$ and $(\text{axis}::a, v) \in \alpha(u)$,
- $s = \text{axis}::[\,exp\,]$ and there exists b such that $(\text{axis}::b, v) \in \alpha(u)$ and $exp \in \text{label}(b)$,

where axis $\in$ {self, next, next^{-1}, edge, edge^{-1}, node, node^{-1}}. Additionally, if $q \in \delta(p, \varepsilon)$ we have that $(u, q) \in \delta'((u, p), \varepsilon)$ for every $u \in \text{voc}(G)$. That is, $G \times \mathcal{A}_{exp}$ is the standard product automaton of G and $\mathcal{A}_{exp}$ if G is viewed as an ε-NFA over the alphabet $\mathbf{D}_0(exp)$. Figure 6 shows the product automaton $G \times \mathcal{A}_\beta$ for the nested expression β in ([11]) and the graph G of Fig. 5 (labeled with respect to the nested subexpressions of β). In this figure, we have only depicted the states of $G \times \mathcal{A}_\beta$ that are reachable from the initial state. For instance, we have that there is a transition from (r_2, q_1) to (r_4, q_1) with symbol next::[next::b/self::c] since: (i) there is a transition from q_1 to q_1 with next::[next::b/self::c] in $\mathcal{A}_\beta$, and (ii) $(\texttt{next}::r_3, r_4) \in \alpha(r_2)$ and $\gamma = \texttt{next}::b/\texttt{self}::c \in \text{label}(r_3)$.

Two key observations about the product automaton defined above should be made. Let G be a graph labeled with respect to the nested subexpressions of exp, and $\mathcal{A}_{exp}$ an ε-NFA for exp. Assume that q_0 is the initial state of $\mathcal{A}_{exp}$ and q_f is one of its final states. The first observation is that if there exists two elements $u, v \in \text{voc}(G)$ such that from (u, q_0) one can reach state (v, q_f) in $G \times \mathcal{A}_{exp}$, then $(u, v) \in [\![exp]\!]_G$. In the example of Fig. 6, we have that $(r_1, r_5) \in [\![\beta]\!]_G$ since we can reach state (r_5, q_f) from state (r_1, q_0) in $G \times \mathcal{A}_\beta$. The second observation is that given a nested regular expression *exp*, one can construct in linear time an ε-NFA for *exp* by using standard techniques [28]. Thus, given a nested regular expression *exp* and an RDF graph G that has been labeled with respect to the nested subexpressions of *exp*, it is easy to see that automaton $G \times \mathcal{A}_{exp}$ can be constructed in time $O(|G| \cdot |\mathcal{A}_{exp}|)$.

Now we have all the necessary ingredients to present the algorithm for the evaluation problem for nested regular expressions given in [43]. This algorithm is split in two procedures: LABEL labels G according to the nested subexpressions of *exp* as explained above, and EVAL returns YES if $(a, b) \in [\![exp]\!]_G$ and NO otherwise.

```
LABEL($G$, $exp$):
1. for each axis::[$exp'$] $\in \mathbf{D}_0(exp)$ do
2.      call LABEL($G$, $exp'$)
3. construct $\mathcal{A}_{exp}$, and assume that $q_0$ is its initial state and $F$ is its set of final states
4. construct $G \times \mathcal{A}_{exp}$
5. for each $(u, q_0)$ that is connected to a state $(v, q_f)$ in $G \times \mathcal{A}_{exp}$, with $q_f \in F$ do
6.      label($u$) := label($u$) $\cup \{exp\}$
```

```
EVAL($G$, $exp$, $(a, b)$):
1. for each $u \in$ voc($G$) do
2.      label($u$) := $\emptyset$
3. call LABEL($G$, $exp$)
4. construct $\mathcal{A}_{exp}$, and assume that $q_0$ is its initial state and $F$ is its set of final states
5. construct $G \times \mathcal{A}_{exp}$
6. if a state $(b, q_f)$, with $q_f \in F$, is reachable from $(a, q_0)$ in $G \times \mathcal{A}_{exp}$
7.      then return YES
8.      else return NO
```

In [43], it is formally proved that procedure EVAL can be implemented efficiently. More precisely, assuming that $|exp|$ denotes the size of a nested regular expression *exp*, it is shown in [43] that:

Theorem 12 ([43]). *Procedure* EVAL *solves the evaluation problem for nested regular expressions in time* $O(|G| \cdot |exp|)$.

6.3 The Navigational Language nSPARQL

We conclude this section by presenting the query language nSPARQL introduced in [43], and showing that the navigational capabilities of nSPARQL can be used to pose many interesting and natural queries over RDF data. In particular, we formally show that these capabilities can be used to evaluate queries according to the semantics of the RDFS vocabulary.

The language nSPARQL (*nested* SPARQL) is obtained by using triple patterns with nested regular expressions in the predicate position, plus SPARQL operators AND, OPT, UNION, and FILTER. Formally, a *nested-regular-expression triple* (or just nre-triple) is a tuple t of the form (x, exp, y), where $x, y \in U \cup V$ and *exp* is a nested regular expression. nSPARQL patterns are recursively defined from nre-triples:

– An nre-triple is an nSPARQL pattern.
– If P_1 and P_2 are nSPARQL patterns and R is a built-in condition, then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, $(P_1 \text{ UNION } P_2)$, and $(P_1 \text{ FILTER } R)$ are nSPARQL patterns.

To define the semantics of nSPARQL, we just need to define the semantics of nre-triples. The evaluation of an nre-triple $t = (?X, \mathit{exp}, ?Y)$ over an RDF graph G is defined as the following set of mappings:

$$[\![t]\!]_G = \{\mu \mid \text{dom}(\mu) = \{?X, ?Y\} \text{ and } (\mu(?X), \mu(?Y)) \in [\![\mathit{exp}]\!]_G\}.$$

Similarly, the evaluation of an nre-triple $t = (?X, \mathit{exp}, a)$ over an RDF graph G, where $a \in U$, is defined as $\{\mu \mid \text{dom}(\mu) = \{?X\} \text{ and } (\mu(?X), a) \in [\![\mathit{exp}]\!]_G\}$, and likewise for $(a, \mathit{exp}, ?X)$ and (a, exp, b) with $b \in U$.

Notice that every SPARQL triple $(?X, p, ?Y)$ with $p \in U$ is equivalent to nSPARQL triple $(?X, \texttt{next}{::}p, ?Y)$. Also notice that, since variables are not allowed in nested regular expressions, the occurrence of variables in the predicate position of triple patterns is forbidden in nSPARQL. Nevertheless, every SPARQL triple of the form $(?X, ?Y, a)$, with $a \in U$, is equivalent to nSPARQL pattern $(?X, \texttt{edge}{::}a, ?Y)$, and every triple of the form $(a, ?X, ?Y)$ is equivalent to $(?X, \texttt{node}{::}a, ?Y)$. Thus, what one loses in nSPARQL is only the possibility of using variables in the three positions of a triple pattern [43].

In the following examples, we show that the navigational capabilities of nSPARQL can be used to pose queries that are likely to occur in the Semantic Web, and which cannot be expressed in SPARQL without using nested regular expressions.

Example 8. Assume that we want to obtain the pairs of cities $(?X, ?Y)$ such that there is a way to travel from $?X$ to $?Y$ by using either Seafrance or NExpress, with an intermediate stop in a city that has a direct NExpress trip to London. Consider nested expression:

$$\begin{aligned}\mathit{exp}_1 = (\texttt{next}{::}\text{Seafrance} \mid \texttt{next}{::}\text{NExpress})^+/ \\ \texttt{self}{::}[\texttt{next}{::}\text{NExpress}/\texttt{self}{::}\text{London}]/ \\ (\texttt{next}{::}\text{Seafrance} \mid \texttt{next}{::}\text{NExpress})^+\end{aligned}$$

Then pattern $P = (?X, \mathit{exp}_1, ?Y)$ answers our initial query. Notice that expression $\texttt{self}{::}[\texttt{next}{::}\text{NExpress}/\texttt{self}{::}\text{London}]$ is used to perform the intermediate existential test of having a direct NExpress trip to London. □

Example 9. Let G be the graph in Fig. 3 and P_1 the following pattern:

$$P_1 = (?X,\ \texttt{next}{::}[(\texttt{next}{::}\texttt{sp})^*/\texttt{self}{::}\text{transport}],\ ?Y). \tag{12}$$

Pattern P_1 defines the pairs of cities $(?X, ?Y)$ such that, there exists a triple $(?X, p, ?Y)$ in the graph and a path from p to transport where every edge has label sp. Thus, nested expression $[(\texttt{next}{::}\texttt{sp})^*/\texttt{self}{::}\text{transport}]$ is used to emulate

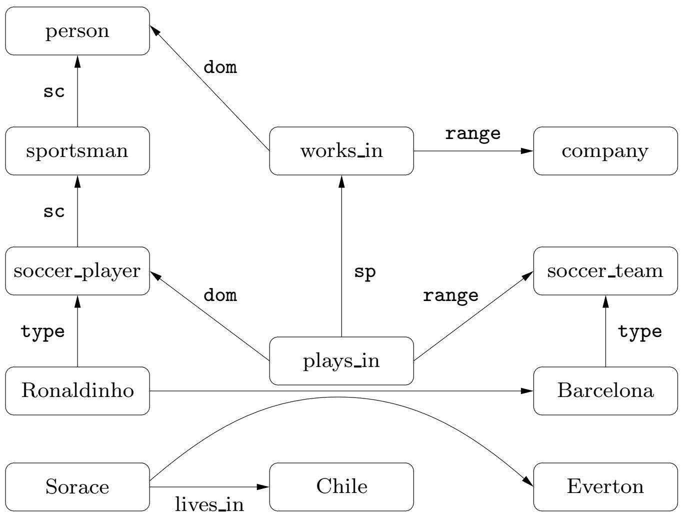


Fig. 7. An RDF graph storing information about soccer players

the process of inference in RDFS; it retrieves all the nodes that are *sub-properties* of transport. Hence, pattern P_1 retrieves the pairs of cities that are connected by a direct transportation service, which could be a train, ferry, bus, etc. In general, if we want to obtain the pairs of cities such that there is a way to travel from one city to the other, we can use the following nSPARQL pattern:

$$P_2 = (?X,\ (\texttt{next}::[(\texttt{next}::\texttt{sp})^*/\texttt{self}::\text{transport}])^+,\ ?Y). \tag{13}$$

□

The previous example shows that nSPARQL can be used to emulate some of the inference rules of RDFS. In [43], it is shown that this is not a particular phenomenon, that is, it is formally proved that if one wants to answer a SPARQL query P according to the semantics of RDFS, then one can rewrite P into an nSPARQL query Q such that Q retrieves the answer to P by directly traversing the input graph. In the remaining of this section, we show how this is done.

SPARQL follows a *subgraph-matching* approach, and thus, a SPARQL query treats RDFS vocabulary without considering its predefined semantics. For example, consider the RDF graph G in Fig. 7, which stores information about soccer players, and consider the graph pattern $P = (?X, \text{works\_in}, ?C)$. Note that, although the triples (Ronaldinho, works_in, Barcelona) and (Sorace, works_in, Everton) can be *deduced* from G, one obtains the empty set as the result of evaluating P over G as there is no triple in G with "works_in" in the predicate position.

We are interested in defining the semantics of SPARQL over RDFS, that is, taking into account not only the explicit RDF triples of a graph G, but also

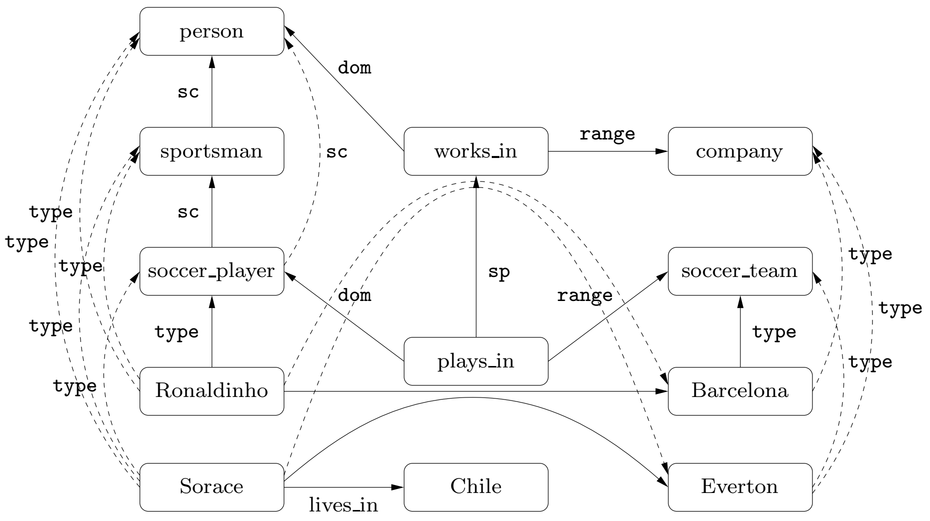


Fig. 8. The closure of the RDF graph of Fig. 7

the triples that can be derived from G according to the semantics of RDFS. We make an initial restriction. In the rest of the paper we assume that RDFS vocabulary cannot occur in subject or object position in RDF triples. Supported on Proposition 1 (2), we only consider rules (2)-(4) for the semantics of RDFS. Let the *closure* of an RDF graph G, denoted by $\mathrm{cl}(G)$, be the graph obtained from G by successively applying rules (2)-(4) in Tab. 1 until the graph does not change. For instance, Fig. 8 shows the closure of the RDF graph of Fig. 7. The solid lines in Fig. 8 represent the triples in the original graph, and the dashed lines the additional triples in the closure.

The most direct way to define the semantics of the RDFS evaluation of SPARQL patterns is by considering not the original graph but its closure. Thus, if we now evaluate pattern $P = (?X, \text{works\_in}, ?C)$ over the RDF graph in Fig. 8, we obtain the mappings $\{?X \to \text{Ronaldinho}, ?C \to \text{Barcelona}\}$ and $\{?X \to \text{Sorace}, ?C \to \text{Everton}\}$. The theoretical formalization of such an approach was studied in [23]. The following definition formalizes this notion.

Definition 6 ([43]). *Given a* SPARQL *graph pattern P, the* RDFS evaluation of P over G*, denoted by $[\![P]\!]^{\mathrm{rdfs}}_G$, is defined as the set of mappings $[\![P]\!]_{\mathrm{cl}(G)}$, that is, as the evaluation of P over the closure of G.*

Let us show with an example how nSPARQL can be used to obtain the RDFS evaluation of some patterns by directly traversing the input graph.

Example 10. Let G be the RDF graph in Fig. 7, and assume that we want to obtain the *type* information of Ronaldinho. This information can be obtained by computing the RDFS evaluation of the pattern $(\text{Ronaldinho}, \texttt{type}, ?C)$. By simply inspecting the closure of G in Fig. 8, we obtain that the RDFS evaluation of (Ronaldinho, $\texttt{type}$, $?C$) is the set of mappings:

$$\{\{?C \rightarrow \text{soccer\_player}\}, \{?C \rightarrow \text{sportsman}\}, \{?C \rightarrow \text{person}\}\}.$$

However, if we directly evaluate this pattern over G, we obtain a single mapping $\{?C \rightarrow \text{soccer\_player}\}$. Consider now the nSPARQL pattern:

$$P = (\text{Ronaldinho},\ \texttt{next::type/(next::sc)}^*,\ ?C).$$

The expression $\texttt{next::type/(next::sc)}^*$ is intended to obtain the pairs of nodes such that there is a path between them that starts with label `type` followed by zero or more labels `sc`. When evaluating this expression in G, we obtain the set of pairs {(Ronaldinho, soccer_player), (Ronaldinho, sportsman), (Ronaldinho, person), (Barcelona, soccer_team)}. Thus, the evaluation of P results in the set of mappings:

$$\{\{?C \rightarrow \text{soccer\_player}\}, \{?C \rightarrow \text{sportsman}\}, \{?C \rightarrow \text{person}\}\}.$$

In this case, pattern P is enough to obtain the type information of Ronaldinho in G according to the RDFS semantics, that is,

$$[\![(\text{Ronaldinho}, \texttt{type}, ?C)]\!]_G^{\text{rdfs}} = [\![(\text{Ronaldinho},\ \texttt{next::type/(next::sc)}^*,\ ?C)]\!]_G.$$

Although the expression $\texttt{next::type/(next::sc)}^*$ is enough to obtain the type information for Ronaldinho in G, it cannot be used in general to obtain the type information of a resource. For instance, in the same graph, assume that we want to obtain the type information of Everton. In this case, if we evaluate the pattern (Everton, $\texttt{next::type/(next::sc)}^*$, $?C$) over G, we obtain the empty set. Consider now the nSPARQL pattern:

$$Q = (\text{Everton},\ \texttt{node}^{-1}\texttt{/(next::sp)}^*\texttt{/next::range},\ ?C).$$

With the expression $\texttt{node}^{-1}\texttt{/(next::sp)}^*\texttt{/next::range}$, we follow a path that first navigates from a node to one of its incoming edges by using $\texttt{node}^{-1}$, and then continues with zero or more `sp` edges and a final `range` edge. The evaluation of this expression over G results in the set {(Everton, soccer_team), (Everton, company), (Barcelona, soccer_team), (Barcelona, company)}. Thus, the evaluation of Q in G is the set of mappings:

$$\{\{?C \rightarrow \text{soccer\_team}\}, \{?C \rightarrow \text{company}\}\}.$$

By looking at the closure of G in Fig. 8, we see that pattern Q obtains exactly the type information of Everton in G, that is, $[\![(\text{Everton}, \texttt{type}, ?C)]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$.
□

Next we show how the ideas in Examples 9 and 10 were generalized in [43] to obtain a way to evaluate a SPARQL query according to the RDFS semantics. More precisely, we show that if a SPARQL pattern P is constructed by using triple patterns having at least one position with a non-variable element, then the RDFS evaluation of P can be obtained by directly traversing the input graph with an nSPARQL pattern.

Consider the following *translation* function from elements in U to nested regular expressions:

$$\begin{array}{ll}
\mathit{trans}(\texttt{sc}) & = (\texttt{next::sc})^+ \\
\mathit{trans}(\texttt{sp}) & = (\texttt{next::sp})^+ \\
\mathit{trans}(\texttt{dom}) & = \texttt{next::dom} \\
\mathit{trans}(\texttt{range}) & = \texttt{next::range} \\
\mathit{trans}(\texttt{type}) & = (\ \texttt{next::type}/(\texttt{next::sc})^* \mid \\
 & \quad\ \texttt{edge}/(\texttt{next::sp})^*/\texttt{next::dom}/(\texttt{next::sc})^* \mid \\
 & \quad\ \texttt{node}^{-1}/(\texttt{next::sp})^*/\texttt{next::range}/(\texttt{next::sc})^*\ ) \\
\mathit{trans}(p) & = \texttt{next::}[(\texttt{next::sp})^*/\texttt{self::}p] \ \text{ for } p \notin \{\texttt{sc}, \texttt{sp}, \texttt{range}, \texttt{dom}, \texttt{type}\}.
\end{array}$$

Notice that this translation function has been implicitly used in Examples 9 and 10. In the following lemma, it is shown that given an RDF graph G and a triple pattern t not containing a variable in the predicate position, the above translation function can be used to obtain the RDFS evaluation of t over G by navigating G through a nested regular expression.

Lemma 3 ([43]). *Let (x, p, y) be a* SPARQL *triple pattern with $x, y \in U \cup V$ and $p \in U$. Then $[\![(x, p, y)]\!]_G^{\mathrm{rdfs}} = [\![(x, \mathit{trans}(p), y)]\!]_G$ for every RDF graph G.*

Suppose now that we have a SPARQL triple pattern t with a variable in the predicate position, but such that the subject and object of t are not both variables. Next it is shown how to construct an nSPARQL pattern P_t such that $[\![t]\!]_G^{\mathrm{rdfs}} = [\![P_t]\!]_G$ [43]. Assume that $t = (x, ?Y, a)$ with $x \in U \cup V$, $?Y \in V$, and $a \in U$, that is, t does not contain a variable in the object position. Consider for every $p \in \{\texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}, \texttt{type}\}$, the pattern $P_{t,p}$ defined as:

$$((x, \mathit{trans}(p), a) \text{ AND } (?Y, \texttt{self::}p, ?Y)).$$

Then define pattern P_t as follows:

$$P_t = ((x, \texttt{edge::}a/(\texttt{next::sp})^*, ?Y) \text{ UNION } P_{t,\texttt{sc}} \text{ UNION } P_{t,\texttt{sp}} \text{ UNION } P_{t,\texttt{dom}} \text{ UNION } P_{t,\texttt{range}} \text{ UNION } P_{t,\texttt{type}}).$$

In a similar way, it is possible to define pattern P_t for a triple pattern $t = (a, ?Y, x)$, where $a \in U$, $?Y \in V$ and $x \in U \cup V$. By using this construction, it is shown in [43] that:

Lemma 4 ([43]). *Let $t = (x, ?Y, z)$ be a triple pattern such that $?Y \in V$ and $x \notin V$ or $z \notin V$. Then $[\![t]\!]_G^{\mathrm{rdfs}} = [\![P_t]\!]_G$ for every RDF graph G.*

Let $\mathcal{T}$ be the set of triple patterns of the form (x, y, z) such that $x \notin V$ or $y \notin V$ or $z \notin V$. We have shown how to translate every triple pattern $t \in \mathcal{T}$ into an nSPARQL pattern P_t such that $[\![t]\!]_G^{\mathrm{rdfs}} = [\![P_t]\!]_G$. Moreover, for every triple pattern t, its translation is of size linear in the size of t. Given that the semantics of SPARQL is defined from the evaluation of triple patterns, the following results follows:

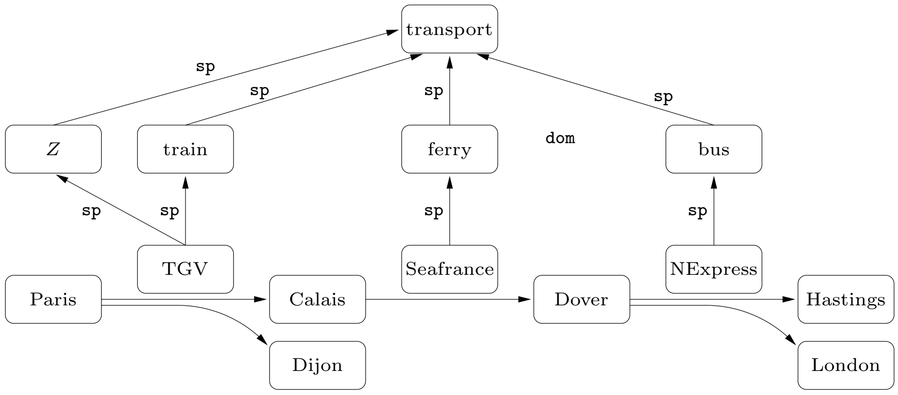


Fig. 9. An RDF graph with RDFS vocabulary and blank nodes

Theorem 13 ([43]). *Let P be a* SPARQL *pattern constructed from triple patterns in $\mathcal{T}$. Then there exists an* nSPARQL *pattern Q such that $[\![P]\!]_G^{\mathrm{rdfs}} = [\![Q]\!]_G$ for every RDF graph G. Moreover, the size of Q is linear in the size of P.*

7 Future Work: Dealing with Blank Nodes

Blank nodes, that is, existential objects, are not new in the area of databases [29,52]. And not only that, they have also been present in the RDF data model since the beginning of the Semantic Web initiative [34]. However, the design of SPARQL was made to keep the efficiency of the language and, in this direction, the current definition of this language does not consider the semantics of blank nodes recommended by the W3C [27]. To see why this is the case, let G_1 and G_2 be the RDF graphs in Figures 3 and 9, respectively, and assume that node Z in G_2 is a blank node. Consider the following SPARQL query:

$$P = \Bigg(\Bigg((?X, \mathtt{sp}, ?V) \text{ AND } (?V, \mathtt{sp}, ?Y) \text{ AND } (?X, \mathtt{sp}, ?W) \text{ AND } (?W, \mathtt{sp}, ?Y)\Bigg) \text{ FILTER} \neg(?V = ?W)\Bigg).$$

Query P evaluated over an RDF graph G retrieves mappings $\{?X \to a, ?Y \to b, ?V \to c, ?W \to d\}$ such that $(a, \mathtt{sp}, c)$, $(c, \mathtt{sp}, b)$, $(a, \mathtt{sp}, d)$ and $(d, \mathtt{sp}, b)$ are all triples in G and c, d are distinct elements. Notice that the clause FILTER $\neg(?V = ?W)$ is used to indicate that $?V$ and $?W$ must take distinct values. Under the W3C semantics for blank nodes [27], G_1 and G_2 are equivalent as blank node Z in G_2 can be identified with node `train`. Therefore, one would expect that the answer to P over G_1 is the same as over G_2. However, this is not the case; Z and `train` are considered to be distinct values under the semantics for SPARQL proposed in [45] and, thus, mapping

$\{?X \rightarrow \text{TGV}, ?Y \rightarrow \text{transport}, ?V \rightarrow \text{train}, ?W \rightarrow Z\}$ is in the answer of P over G_2 but not in the answer of P over G_1.

Evaluating queries which involve blank nodes is challenging, and there is not yet consensus in the Semantic Web community on how to define a query language for this type of data. As an important problem for future work, we identify the issue of extending SPARQL to consider RDF data with blank nodes. In practice, a considerable number of RDF databases include this type of nodes and, thus, this project is driven by the need to extend SPARQL to cope with this data. We hope that a project like this will help in bridging the gap between the current specification of SPARQL [45] and both the definition of the semantics of RDF data [27] and the way RDF data is used in real life.

We conclude this section by pointing out that blank nodes are used not only on RDF graphs but also in SPARQL patterns. They were introduced to make SPARQL compatible with future logical extensions. Nevertheless, they play no major role in the current semantics. In fact, it can be shown that each SPARQL query Q can be simulated by a SPARQL query Q' not mentioning any blank nodes. More precisely, it follows from the definitions of RDF instance mapping, solution mapping, and the order of evaluation of solution modifiers (see [45]), that if Q' is obtained from Q by replacing each blank node B by a fresh variable $?X_B$, then Q and Q' give the same results.

Acknowledgments

This paper is a survey of well-known results on the theory of RDF, which compiles and summarizes results of papers of the authors and their colleagues Renzo Angles, Carlos Hurtado, Alberto Mendelzon and Sergio Muñoz. The authors were supported by: Arenas - Fondecyt grant 1090565; Gutierrez - Fondecyt grant 1070348; Pérez - Conicyt Ph.D. Scholarship; Arenas, Gutierrez and Pérez - grant P04-067-F from the Millennium Nucleus Center for Web Research.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Alechina, N., Immerman, N.: Reachability Logic: An Efficient Fragment of Transitive Closure Logic. Logic Journal of the IGPL 8(3), 325–338 (2000)
3. Alkhateeb, F.: Querying RDF(S) with Regular Expressions. PhD Thesis, Université Joseph Fourier, Grenoble, FR (2008)
4. Alkhateeb, F., Baget, J., Euzenat, J.: RDF with regular expressions. Research Report 6191, INRIA (2007)
5. Alkhateeb, F., Baget, J., Euzenat, J.: Constrained regular expressions in SPARQL. In: SWWS 2008, pp. 91–99 (2008)
6. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. 40(1), 1–39 (2008)
7. Angles, R., Gutierrez, C.: The Expressive Power of SPARQL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008)

8. Anyanwu, K., Maduko, A., Sheth, A.: SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases. In: WWW 2007, pp. 797–806 (2007)
9. Arenas, M., Gutierrez, C., Parsia, B., Pérez, J., Polleres, A., Seaborne, A.: SPARQL - Where are we? Current state, theory and practice. Unit-2: SPARQL Formalization. In: Tutorial given at ESWC 2007, Innsbruck, Austria (2007), `http://axel.deri.ie/~axepol/sparqltutorial/`
10. Arenas, M., Gutierrez, C., Pérez, J.: An Extension of SPARQL for RDFS. In: Christophides, V., Collard, M., Gutierrez, C. (eds.) SWDB-ODBIS 2007. LNCS, vol. 5005, pp. 1–20. Springer, Heidelberg (2008)
11. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (February 2004), `http://www.w3.org/TR/rdf-schema/`
12. Benedikt, M., Koch, C.: XPath leashed. ACM Computing Surveys 41(1) (2008)
13. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002)
14. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Rewriting of Regular Expressions and Regular Path Queries. J. Comput. Syst. Sci (JCSS) 64(3), 443–465 (2002)
15. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. Journal of Web Semantics 3, 247–267 (2005)
16. Chandra, A.K., Merlin, P.M.: Optimal Implementation of Conjunctive Queries in Relational Data Bases. In: STOC 1977, pp. 77–90 (1977)
17. Clark, J., DeRose, S.: XML Path Language (XPath). W3C Recommendation (November 1999), `http://www.w3.org/TR/xpath`
18. Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge (2000)
19. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. McGraw-Hill, New York (2003)
20. Cyganiak, R.: A relational algebra for SPARQL. Tech. Rep. HPL-2005-170, HP-Labs (2005), `http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html`
21. Galindo-Legaria, C.A., Rosenthal, A.: Outerjoin simplification and reordering for query optimization. TODS 22(1), 43–73 (1997)
22. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
23. Gutierrez, C., Hurtado, C., Mendelzon, A.: Foundations of Semantic Web Databases. In: PODS 2004, pp. 95–106 (2004)
24. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
25. Harris, S., Gibbins, N.: 3store: Efficient bulk RDF storage. In: PSSS 2003, pp. 1–15 (2003)
26. Hayes, J., Gutierrez, C.: Bipartite Graphs as Intermediate Model for RDF. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 47–61. Springer, Heidelberg (2004)
27. Hayes, P.: RDF Semantics. W3C Recommendation (February 2004), `http://www.w3.org/TR/rdf-mt/`
28. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Reading (2006)
29. Imielinski, T., Lipski Jr., W.: Incomplete Information in Relational Databases. J. ACM 31(4), 761–791 (1984)
30. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: a declarative query language for RDF. In: WWW 2002, pp. 592–603 (2002)

31. Kochut, K.J., Janik, M.: SPARQLeR: Extended Sparql for Semantic Association Discovery. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 145–159. Springer, Heidelberg (2007)
32. Lassila, O., Swick, R.: Resource description framework (RDF) model and syntax specification W3C Recommendation (February 1999), `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`
33. Levene, M., Loizou, G.: A Guided Tour of Relational Databases and Beyond. Springer, Heidelberg (1999)
34. Manola, F., Miller, E., McBride, B.: RDF Primer, W3C Recommendation (February 10 , 2004), `http://www.w3.org/TR/REC-rdf-syntax/`
35. Marin, D.: RDF Formalization, Santiago de Chile, Technical Report Universidad de Chile, TR/DCC-2006-8 (2004), `http://www.dcc.uchile.cl/~cgutierr/ftp/draltan.pdf`
36. Mendelzon, A., Wood, P.: Finding Regular Simple Paths in Graph Databases. SIAM J. Comput. 24(6), 1235–1258 (1995)
37. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal Deductive Systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 53–67. Springer, Heidelberg (2007)
38. Olson, M., Ogbuji, U.: The Versa Specification, `http://uche.ogbuji.net/tech/rdf/versa/etc/versa-1.0.xml`
39. ODP - Open Directory Project, `http://www.dmoz.org/`
40. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006)
41. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL (submitted for publication)
42. Pérez, J., Arenas, M., Gutierrez, C.: Semantics of SPARQL. Tech. Report Universidad de Chile, TR/DCC-2006-17 (2006)
43. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A Navigational Language for RDF. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 66–81. Springer, Heidelberg (2008)
44. Polleres, A.: From SPARQL to rules (and back). In: Proceedings of the 16th International World Wide Web Conference (WWW), pp. 787–796. ACM, New York (2007)
45. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
46. RDF Site Summary (RSS) 1.0, `http://web.resource.org/rss/1.0/`
47. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL Query Optimization. arXiv.org paper arXiv:0812.3788v1 (December 19, 2008)
48. The Dublin Core Metadata Initiative, `http://dublincore.org/`
49. The Friend of a Friend (FOAF) project, `http://www.foaf-project.org/`
50. Uniform Resource Identifier (URI): Generic Syntax, `http://tools.ietf.org/html/rfc3986`
51. Vardi, M.Y.: The Complexity of Relational Query Languages (Extended Abstract). In: STOC 1982, pp. 137–146 (1982)
52. Zaniolo, C.: Database Relations with Null Values. J. Comput. Syst. Sci. 28(1), 142–166 (1984)

Database Technologies for RDF

Souripriya Das and Jagannathan Srinivasan

Oracle,
1 Oracle Dr, Nashua, NH, USA
{Souripriya.Das,Jagannathan.Srinivasan}@Oracle.com

Abstract. Efficient and scalable support for RDF/OWL data storage, loading, inferencing and querying, in conjunction with already available support for enterprise level data and operations reliability requirements, can make databases suitable to act as enterprise-level RDF/OWL repository and hence become a viable platform for building semantic applications for the enterprise environments.

This tutorial outlines the requirements for supporting semantic technologies in databases including bulk load and data manipulation operations, inference based on RDFS, OWL and user-defined rules, and support for SPARQL queries. It also discusses the design choices for handling issues that arise in implementing support for storage and operations on large scale RDF/OWL data, and in general, touches upon the practical aspects related to RDF/OWL support that become important in enterprise environments. Semantic technologies support in Oracle Database is used as a case study to illustrate with concrete examples the key requirements and design issues.

Keywords: data characteristics, storage architecture, bulk load, inference, semantic query, ontology-assisted query, programming interface, performance, query hints, indexing.

1 Introduction

Resource Description Framework (RDF) [1], the W3C standard for representing metadata about resources, is being adopted widely especially for building semantic applications. Also, the related W3C standards RDFS [2] and OWL [3], which provide richer constructs to represent thesaurus, ontology, and knowledge, in general, are also gaining popularity. For querying, W3C has recommended SPARQL [10], which allows graph pattern based querying over RDF(S)/OWL datasets.

The creation of these standards and their adoption is setting the stage for utilizing semantic technologies in enterprises. For enterprise level use of semantic technologies the key requirements are performance, scalability, reliability, and robustness for loading data, inferencing, and queries. There are some obstacles as well such as demonstrating the benefits of semantic technologies and also showing that the efforts need to build semantic applications is reasonable. Furthermore, creating or obtaining custom RDF data and well-designed RDFS/OWL ontologies, techniques for integrating data from multiple sources and still be able to use inference and query in a meaningful way is challenging. Another challenge in building semantic applications is that businesses typically need to stay ahead of its competition in terms of increased productivity and fastest way to get to the market with products that closely match the requirements of the customers. Thus, there is need for tools to facilitate development of semantic applications.

However, in this tutorial paper we focus on use of DBMS as a platform for building semantic applications in enterprises. The choice of DBMS is driven by its proven record over decades with respect to performance, scalability, security, and transactional support. Using DBMS to manage RDF (semantic repository) entails database schema design, examining and adopting traditional bulk loading, incremental loading, and querying capabilities to suite RDF needs. In addition, graph pattern based querying over RDF data needs to be supported. Another novel aspect is the need to support inferencing (derivation of new triples) from asserted RDF facts based on semantics of RDF(S) and OWL constructs. All these aspects are discussed in the remainder of the paper. Also, to provide reader with concrete examples, we present a case study of design and implementation of semantic store using Oracle Database.

The rest of the paper is organized as follows: Section 2 discusses the key aspects of a semantic store implemented in a database. Section 3 presents the case study of semantic store implemented using Oracle Database. Section 4 summarizes the paper.

2 Key Concepts

The key aspects of a semantic store implemented in a database are its: 1) storage architecture, and capabilities for 2) Loading, 3) Inference, and 4) Query.

2.1 Storage Architecture

Schema-oblivious vs. Schema-aware. The classification of storage schemes described here is based mainly on [23]. The storage for RDF triples could consist of a single three-column *triple* table, where the columns correspond to the subject, predicate, and object components of RDF triples. This is sometimes classified as a schema-oblivious approach because the storage table does not change even if the schema of the RDF data to be stored changes.

Unlike a schema-oblivious approach, a schema-aware approach (such as the one described in [21]) involves choosing a set of tables based upon the schema of the RDF data to be stored. For example, a storage scheme could have a separate two-column *property* table for each different predicate used in the dataset for storing the <subject, object> pairs that are related by the property, and a separate single-column *class* table for each of the classes in the RDF schema to store the instances belonging to a class. There are several benefits of using schema-aware approach, such as compact storage of data by avoiding repetition of common values (such as predicate component by using a property table and *rdf:type <className>* portion of RDF triples by using class tables) and more importantly, processing queries by accessing multiple smaller tables instead of a single large table. A major drawback, however, is the fact that the set of tables may need to change as the data evolves especially with insertion of triples containing new predicates, or deletion of all triples containing a particular predicate. Furthermore, processing of SPARQL queries with variables in predicate position (e.g., ?x ?p ?y) or class positions (e.g., ?x rdf:type ?y) would require making provision for querying all the property tables or class tables.

There is another possible approach that could be classified as schema-oblivious. This is slightly different from the hybrid scheme described in [23]. Here, instead of a single three-column table, one may create a predetermined number of tables or a single table with a predetermined number of columns based upon the possible data types of values of object components of triples – one for each recognized built-in data types in XML Schema [24], such as xsd:string, xsd:decimal, xsd:dateTime (possibly with a flag to distinguish xsd:date, xsd:time, and xsd:dateTime with or without time zone) and one for IRIs and blank nodes, and one for all other types. The triples are stored in the appropriate table or if a single table is used, the object component's value is stored in the appropriate column, based upon the data type of its object component.

Id-based vs. Value-based. Typically, RDF datasets contain repeated occurrences of long IRIs. This can be seen in the data characteristics of the RDF datasets in the existing benchmarks such as LUBM [5] and also in various well-known publicly available RDF datasets [6, 7]. This leads to a choice between a value-based approach where the lexical values are stored directly in the storage tables versus an id-based approach where each of the lexical values are mapped to a unique identifier and those identifiers are stored in the storage tables. The id-based approach requires an additional table for storing the 1-to-1 mapping between lexical values and corresponding identifiers. There are additional benefits of the id-based approach, besides the obvious space benefits. Query and inferencing performance benefits are seen because 1) use of shorter identifiers results in smaller size of the indexes or other auxiliary structures and 2) id-based equality evaluation used for equality-based joins is usually more efficient than lexical value based equality evaluation. A variant of this storage scheme is employed by Jena2 [28], which uses a denormalized schema in which resource URIs and simple literal values are stored directly in the statement (triple) table.

Generation of identifiers in an id-based approach may be done using sequence or using hash functions. Use of powerful but efficient hash functions provides several benefits: 1) there is virtually no dependence on history of insertions or deletions of triples, 2) hash identifiers can be obtained simply by applying the hash function on lexical values and consulting a rarely non-empty collisions table. It may be noted that this choice is relevant irrespective of whether schema-oblivious or schema-aware approach is used.

Prefix compression. Many distinct IRIs used in RDF datasets often have the same prefix. A storage scheme could optionally try to factor out the common prefixes to reduce the space overhead, which also contributes to performance benefits by reducing the size of the table and indexes. There are at least two ways for doing this. One could associate unique identifiers with each distinct prefix and store the mapping in a separate table and then use the combination of the prefix-id and the suffix lexical value to represent a full lexical value. Another possible way is to store the prefix and suffix portions of lexical values in two distinct columns of a table and use table compression. The latter approach avoids having to create a separate table and hence the need to do a join with an extra table, but the extent of compression is limited by the extent to which values sharing the same prefix are co-located in a database block. It may be noted that when id-based triples are used, this prefix compression may be used in the value↔id mapping table.

Ancillary values. Often it becomes necessary to store additional <predicate, object> pairs for an RDF triple, not just for IRIs (or blank nodes). RDF allows storing of such ancillary information via reification by using reification quad to represent a reified RDF triple and then creating triples to associate ancillary <predicate, object> pairs to the subject used in the reification quad. While this approach is very powerful in that it allows unlimited levels of reification (e.g., even the association of <predicate, object> to an RDF triple may be expressed as a reification quad, so that additional <predicate, object> pairs may be associated with the association, and so on), it can potentially increase the number of triples by a factor of four (due to use of quads) and can increase the number of joins needed for query processing. A storage scheme could allow a simpler way of storing ancillary values associated with a triple by extending the set of columns for an RDF triple by adding additional columns to the set {subject, predicate, object}. A common addition is a graph name or context column. Although not as powerful as the reification quad, this approach may be used for adding ancillary information such as security, uncertainty, and so on at the triple level granularity.

Subject	Predicate	Object	Anc. Property1	Anc. Property 2	…

Fig. 1. Storage scheme allowing ancillary values for RDF triples

Unit of storage, ownership, and access control. Since a database may store many RDF graphs, each RDF graph can be considered a unit of storage, ownership, and access control. Thus, when a user creates an RDF graph, possibly empty to start with, a name is associated with the graph and the creator is listed as the owner of the graph. Furthermore, in order to allow the owner to control who can access the graph, owner may be given appropriate privileges with option to grant some of those privileges to other users.

Indexes. Since RDF has a set semantics, duplicate triples are not allowed within a single RDF graph. A uniqueness constraint is defined on the RDF dataset to enforce the set semantics in each graph. For efficient query processing and DML integrity, typically a uniqueness constraint is maintained by use of a unique key index. Besides enforcing the uniqueness constraint, indexes are also used as access path for performance reasons. A storage scheme may create predefined indexes and optionally may provide the flexibility of allowing privileged users to create additional indexes or remove some of the indexes (while maintaining at least one index that enforces the uniqueness constraint). In general, for RDF triple data, six ways of indexing are possible, performance aspects of which is presented in [19]. To keep the space needed for indexes down, a storage scheme may use compression of key prefixes. It may be noted that the retrieval performance benefits of indexes need to be weighed against the performance overhead of maintaining indexes during data loading or other modification operations.

Other Auxiliary Structures. One could also build additional auxiliary structures to speed up query processing such as *subject property matrix* materialized join views [8]. However, like indexes one needs to account for maintenance cost when RDF data is incrementally added. For read-only or read-mostly data, these structures could be beneficial.

RDF view of relational data. In order to allow access to the huge amount of relational data stored in databases, some of the platforms for semantic technologies [25, 26] allow viewing relational data as virtual RDF graphs that can be queries using the W3C SPARQL query language. Typically it includes a mapping language to specify mapping of the tables to classes and columns and constraints to properties and ability to translate a SPARQL query specified against the RDF view of the relational data to SQL queries against the underlying relational tables. [27] surveys the support for RDF views over relational data in existing platforms.

2.2 Loading RDF Data

Incremental load and Bulk load. Usually at least two types of data loading are supported: 1) loading via SQL INSERT statements, typically used for small amount of data, and 2) highly optimized bulk loading APIs to allow efficient loading of large amount of data. Loading data also involves parsing to ensure that values used for subject, predicate, and object components of the triples are indeed valid RDF terms and appropriate for that component. Another important and time-consuming task is elimination of duplicates to maintain the set semantics of each RDF graph. Loading may sometimes include inferencing as well.

Input format. The input for bulk load is usually file-based and formatted in one of the many standard RDF data formats (e.g., N-Triple, RDF/XML, N3, or Turtle). Bulk-load APIs may sometimes also accept input data from a staging table defined as a three-column table with the columns storing the lexical values of the subject, predicate, and object components of the RDF triples.

Effect of storage architecture. A loading scheme used for bulk load is usually closely tied with the storage architecture of the semantic store and also the data characteristics. For example, data loading is straightforward if value-based storage scheme is used, but performance suffers due to the need to write a large number of long lexical values and building corresponding big indexes. Loading scheme for efficient and scalable bulk loading of RDF data into a semantic store that uses the id-based scheme, however, is much more complicated due to the need to load the value↔id mapping table and also the id-based triples table. Use of hash-based identifiers can make bulk loading much more efficient due to the ease of generation of the identifiers corresponding to the lexical values. Similarly, if a semantic store allows storing ancillary values for triples using special structures (instead of using reification quads), then loading gets more complex, and it incurs the overhead of populating those special structures.

Bulk append. Appending (that is, loading into a non-empty RDF graph) requires checking for duplicates within the new batch of triples to be appended and between the new batch and the existing set of triples. Additionally, it may require incremental maintenance of the indexes. Some of these operations make the bulk append operation less efficient than the bulk load operation (loading into empty RDF graph).

Reuse of blank nodes. Appending new triples raises a question about the reuse of blank node names across different batches of triples being loaded. Specifically, if a blank node with a specified label is already present in an RDF graph, and the same blank node label is used in the batch of triples to be appended to the graph, then the question that arises is “do they refer to the same resource”? Reuse of blank node labels has the advantage that even if the dataset is loaded in a single batch or in multiple batches, the net effect is same.

2.3 Inferencing

Inferencing, or computing entailment, is a major attribute of semantic technologies that differentiates it from other relevant technologies. Thus the richness, performance, and scalability of the inference capability are often used as important metrics in evaluating the quality of a semantic store for use with semantic applications.

Standard entailment regimes. There are several standard entailment regimes: semantics of RDF, RDFS, and OWL. Within OWL, there are three dialects – OWL-Lite, OWL-DL, and OWL-Full. Support for RDF and RDFS is simplified by the availability of axioms and rules that represent their semantics. An obstacle to providing support for entailment based on OWL vocabularies has been the lack of ready availability of the axioms and rules for the semantics of OWL dialects. The high computational complexity of supporting entailment with the OWL-Full semantics discourages provision of this capability. Support for major subsets of OWL-Lite and OWL-DL vocabularies have been provided in current semantic stores. It may be noted, however, that in the context of forward chaining based inference engines, any rule that infers triples with new blank nodes (i.e., blank nodes not already present in the data) may lead to explosion in the size of inferred triple set or may even lead to non-termination of inference. To avoid such problems, forward chaining inference engines may often allow users to skip inference rules that may cause data explosion. This aspect also plays a role in arriving at an OWL Lite or OWL-DL subset. For example, instead of supporting an iff semantics for rules, inference is sometimes supported only in one direction (as in pD* [30]) which makes inference sound but incomplete.

Interfacing with third-party reasoners. The ability to interface with third-party reasoners can allow users of a semantic store to avail (more) complete support for inferencing, possibly for subsets of data. For example, complete support for OWL-DL in the Pellet inference engine [4] can be leveraged if a semantic store provides the capability to interface with Pellet. To avoid any scalability issues related to the huge number of instance triples, only the schema triples, which usually are small in numbers compared to instance triples, could be sent to third-party reasoners for inferencing. The resulting fully-entailed schema can then be used in combination with the asserted instance triples for creating entailment locally at the semantic store.

User-defined rules. Since the standard vocabularies cannot handle the full repertoire of custom entailment regime a semantic application may require, it becomes important to provide support for entailment based on arbitrary user-defined rules.

Backward-chaining vs. Forward-chaining. An attribute that differentiates inference capabilities of one semantic store from another is *when* the inferencing is done. The forward-chaining option pre-computes and materializes the inferred triples. The backward-chaining option determines the inferred triples at query processing time and does not materialize the inferred triples for later use.

The pre-computing and materialization used in forward-chaining cause storage space overhead, but can make queries efficient by avoiding inferencing logic at query processing time. However, under certain circumstances, space overhead may become huge. For example, use of transitive, symmetric, and reflexive properties such as owl:sameAs, or use of owl:disjointWith which results in inference of triples with owl:differentFrom for each ordered pair of the instances of the disjoint classes and their respective (disjoint) subclasses, may lead to inferring a large number of triples. Thus, special care needs to be taken in the support for forward-chaining to prevent such situations and provide appropriate alternatives.

Maintenance of pre-computed and materialized triples in a forward-chaining system is another important aspect. Any change in the set of triples or axioms or rules may induce changes in the set of inferred triples. The extent of the induced changes may vary from no changes at all to a huge amount of changes. The quality of a semantic store from a user's perspective may depend on how promptly and efficiently the set of inferred triples can be checked and updated if necessary.

Although a majority of the current semantic stores support forward chaining, recently there have been proposals to restrict the expressivity of OWL to enable answering the queries by simple rewrite mechanisms, as in OWL 2 QL [31].

Explanation and Validation. Since new triples obtained via inferencing may sometimes be completely unexpected, it is important to provide support for creating an explanation consisting of the triples and rules along the paths that leads to the inference of the new triples. Support for validation allows users to check if a set of triples is consistent.

2.4 Querying RDF and Relational Data

The ability to efficiently and easily query RDF graphs in a semantic store is probably the most important aspect from a user's point of view. In particular, when a semantic store is implemented in a database, the question arises whether to use SQL or a standard RDF query language such as SPARQL.

Query Language. Since SQL is a feature-rich declarative query language that has been used in databases for decades and has been optimized for efficient execution, it becomes an obvious candidate for use as the language for querying RDF data. The drawback of asking users to express their query for retrieving information from RDF graphs using SQL is that SQL is not very well suited for specifying graph queries.

An alternative is to support querying in the SPARQL query language [10], which is the W3C recommendation for querying RDF data. The benefit of using SPARQL is that it has been designed to be suitable for querying RDF data. Thus, it is easier to specify a query in SPARQL than in SQL. The drawback of SPARQL, however, is that it is a standalone language that lacks important features that are essential for processing retrieved RDF results. For example, neither aggregate functions nor subqueries are yet allowed in SPARQL specification.

A hybrid of the above two approaches is to allow SPARQL queries to be embedded in SQL where results from a SPARQL query is treated like a table and used as a table data source that can be processed by SQL constructs [8]. This allows the ease of querying RDF data using SPARQL, because the portion of the query that goes against RDF data is expressed in SPARQL. At the same time, it allows further processing the result of the SPARQL portion of the query, possibly even combining the result with relational tables, using the rich constructs of SQL. Other approach is to extend SPARQL with such constructs as in query languages supported in Virtuoso [15], Jena [13], and Sesame [16].

Query performance features. The hybrid approach above could further benefit from a performance point of view if the SPARQL portion of the query can be rewritten as a SQL subquery and optimized together with the rest of the query [8].

A performance related feature would be to include the ability to specify hints, similar to hints used in the SQL implemented by database vendors, for the SPARQL subquery to tune the execution plans for queries, for example, by specifying the order of the join operations. Similarly, for an id-based storage architecture, extensions to SPARQL could allow retrieving only the identifiers whenever possible instead of always retrieving the lexical values thereby avoiding additional costly joins.

Cost-based Query Optimization. The query execution can still leverage the cost-based optimizer typically available as part of database system. However, the graph-pattern based query language requires re-examining the selectivity estimation techniques (e.g. [22]).

Querying relational data in the context of RDF. There are two ways one could query relational and RDF data together. A simple way is to use, in the hybrid query, one or more relational tables to join with the SPARQL subqueries.

Another way would be to treat the values used in a relational table column, possibly augmented to make them like IRIs, as terms used in an RDF graph. A new operator may be defined for use in the WHERE clause of SQL queries to check whether the value of that column in a row (in the relational table) satisfies a SPARQL query (usually making use of that column and) specified as an argument to the operator, and select the row accordingly. The operator acts like a filter. An ancillary operator may be defined to retrieve the mapping(s) for the variables used in the SPARQL query for each row that qualifies [8]. Optionally, the association between a relational table column and RDF graph(s) may be characterized by specifying an integrity constraint [9].

2.5 Miscellanous Aspects

Distributed and/or Federated RDF Stores. The traditional database technique for distribution and replication can also be used to support distributed or federated RDF stores. In fact, the canonical triple model of RDF makes it amenable for doing queries over multiple RDF stores. However, the internal transformation to id-based triples makes it somewhat challenging as the same lexical value may get assigned different id in two stores. Here use of hash-based identifiers such as in Oracle [11] could be beneficial. However, the problem of collision detection and resolution needs to be addressed.

3 Case Study: Oracle Database Semantic Technologies

Many semantic stores (including Jena [13], Oracle [14], Vituoso RDF Triple Store [15], and Sesame [16], C-Store [20]) use database to store and manage RDF data. In this section, we present case study of a semantic store implemented in the Oracle Database.

Support for semantic technologies in Oracle Database enables 1) storage, loading, and manipulation of RDF graphs in Oracle; 2) native inferencing based on RDF graphs and built-in entailment regimes such as RDF/S and major subsets of OWL, and user-defined rules; 3) querying RDF data via a hybrid approach where SQL queries would contain SPARQL graph patterns for looking up the RDF graphs and optionally any inferred triples; 4) querying relational data in the context of RDF data (typically, RDFS or OWL ontologies) via use of SQL operators defined using the Oracle extensibility framework [29].

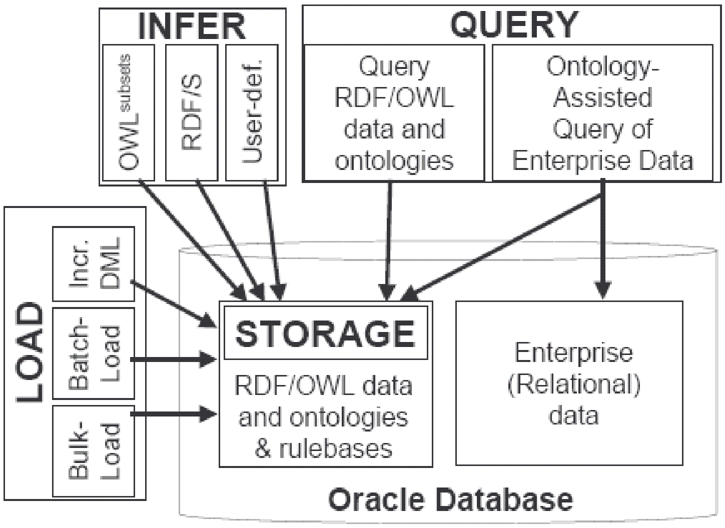

INFER
OWL subsets
RDF/S
User-def.
QUERY
Query RDF/OWL data and ontologies
Ontology-Assisted Query of Enterprise Data
LOAD
Incr. DML
Batch-Load
Bulk-Load
STORAGE
RDF/OWL data and ontologies & rulebases
Enterprise (Relational) data
Oracle Database

Fig. 2. A top-level view of the Oracle Database Semantic Technologies

3.1 Storage Architecture

This section gives an overview of the storage architecture and describes the entities used in an Oracle Database semantic store.

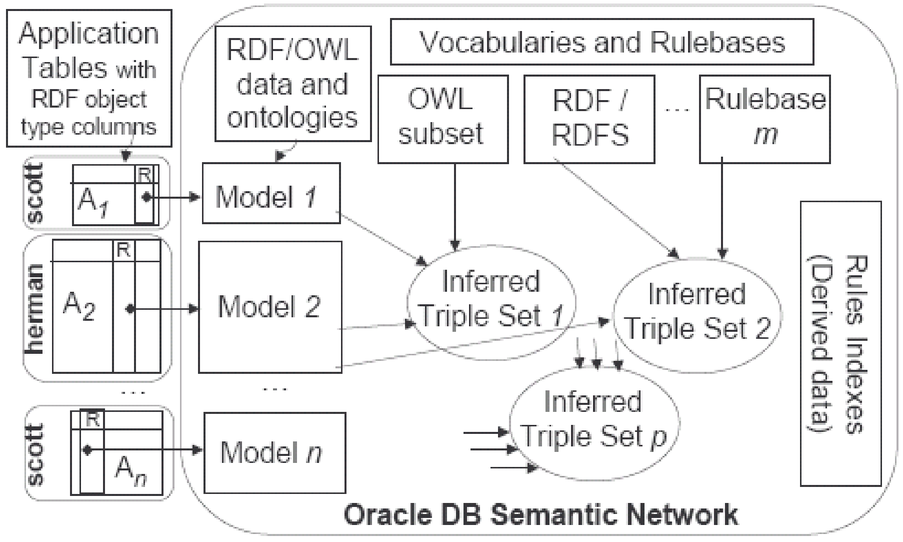

Application Tables with RDF object type columns
scott
A1
R
herman
A2
R
scott
An
R
RDF/OWL data and ontologies
Model 1
Model 2
Model n
Vocabularies and Rulebases
OWL subset
RDF / RDFS
Rulebase m
Inferred Triple Set 1
Inferred Triple Set 2
Inferred Triple Set p
Rules Indexes (Derived data)
Oracle DB Semantic Network

Fig. 3. A top-level view of the entities in an Oracle Database Semantic Network

Overview. Storage architecture of Oracle database semantic store can be characterized as follows:

- Schema-oblivious storage for RDF data: Each RDF model is represented by a database view object. All the view objects for the RDF models have the same definition (i.e., identical set of columns). Access control for the RDF models is provided by leveraging access control support for view objects in Oracle database.
- Id-based: Uses a value↔id mapping table to store one-to-one mapping between lexical values and corresponding identifiers obtained via use of a (native) hash function with complete handling of any rare collisions.
- Prefix compression: Prefix compression is used in the value↔id mapping table. Each IRI into a prefix and suffix, storing the two parts as separate columns. Table compression is used to ensure within a database block, each distinct prefix is stored only once.
- Ancillary values: An additional table, referred to as *application table*, is used for each RDF model to allow storing ancillary values. An application table must have one column of a new object type, SDO_RDF_TRIPLE_S, to store references to id-based RDF triples stored in the corresponding RDF model. Additional columns may be defined in the application table to store ancillary information.

- Unit of storage, ownership, and access control: An RDF model.
- Indexes: Comes with default indexes, including a unique index to enforce the RDF set constraint in each RDF model. Supports adding new indexes and eliminating default indexes (except the unique index used for enforcing the uniqueness constraint). Uses index key-prefix compression for space as well as query performance considerations.

Besides RDF models (described above), the other entities in an Oracle Database semantic network are the following.

- Vocabularies and Rulebases: Semantics for RDF/S and subsets of OWL are built-in in the native inference engine. Users may also create custom rules and store them in rulebases. Each of the rulebases, built-in or user-defined, is represented by a database view object. Access control to rulebases is provided via database view access control, as in the case of RDF models.
- Inferred Triple Sets: New triples generated via entailment on RDF model(s) and using one or more rulebases are stored in units, called *rules indexes*, and modeled as database view objects. Access control to rules indexes is provided via database view access control.

3.2 Loading RDF Data

Loading RDF data into Oracle database semantic store effects the following: 1) any new lexical values to be stored in the value↔id mapping table, 2) each triple is converted to the corresponding id-based form and stored in the table underlying the database view for the target RDF model, and 3) the ancillary values and a key to the row for the triple is stored in the application table.

Overview. Loading RDF data in Oracle database semantic store can be characterized as follows:

- Supports three forms of loading
  - Bulk loading: This method is highly optimized for loading medium to large number (e.g., billions) of triples [17]. The underlying scheme [11] is fairly complex due to the need to map the incoming triples into the id-based storage architecture.
  - Batch loading: This method has been optimized to handle loading a medium number (e.g., a few millions) of triples. An advantage of this method is that, unlike bulk loading, this method does not require object values to stay within 4000 bytes.
  - Loading via SQL INSERT into the application table: This method is recommended for use with small number (e.g., up to a few thousands) of triples.
- Input format: For bulk loading and batch loading, only the N-Triple format file-based input is supported. Bulk loading API also accepts input triples (in prefix-expanded lexical form) from a database table or view (referred to as *staging table*). SQL INSERT requires use of an object type constructor, SDO_RDF_TRIPLE_S, with target RDF model name, and lexical values for subject, predicate, and object components of the triple used as arguments.

- Duplicate elimination during bulk append: When appending a large number of triples into a large RDF model, elimination of duplicates between the new batch of triples and the pre-existing set of triples can be very slow due to use of anti-join. Oracle database semantic store allows use of an optimistic method that can often eliminate the need for anti-join or minimize the anti-join overhead.
- Blank node reuse: All forms of loading assume reuse of blank node labels within an RDF model and no reuse across models.

3.3 Inferencing

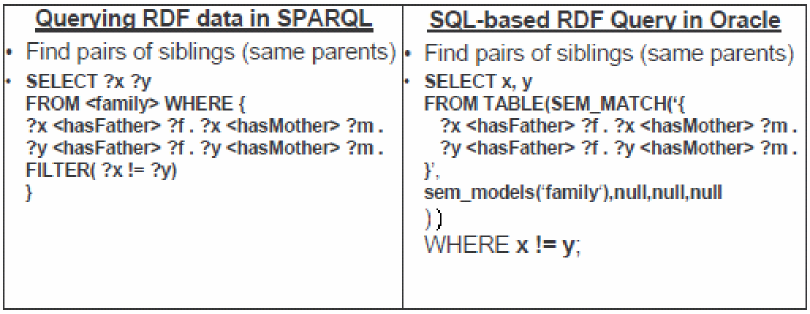
Overview. Inferencing capabilities in Oracle database semantic store can be characterized as follows:

- Native support for standard entailment regimes: A native inference engine supports entailment for RDF, RDFS, and OWLPRIME that is a major subset of OWL-DL.
- Interfacing with third-party reasoners: Through the use of Oracle Jena Adaptor [18], users can optionally make use of the Pellet [4] inference engine for complete OWL-DL inferencing.
- User-defined rules: Supports specification of user-defined rules consisting of an antecedent part, a filter, and a consequent part and entailment using user-defined rules.
- Forward-chaining: The inference engine uses forward-chaining to pre-compute and materialize the inferred triples. The implementation of the inference engine in the database makes heavy use of SQL queries and has been highly optimized to achieve performance and scalability [12] [17].
- Control over choosing components: An entailment regime is presented as consisting of components. Users may choose to include or exclude some of the components when they request creation of entailment. For example, when creating entailment using OWLPRIME, specifying 'SAM-' in the options would cause the inference engine to skip generating new owl:sameAs triples from existing owl:sameAs triples.
- Avoiding explosion in number of inferred triples: To avoid huge increase in number of inferred triples associated sometimes with use of owl:sameAs, owl:disjointWith, etc., the entailment regimes in Oracle have been customized to exclude some of the components by default (but users are allowed to request inclusion of such components, if they choose to, when requesting creation of entailment). If additional component exclusions are needed, users can always specify that option when requesting creation of entailment. New enhancements are currently planned that will avoid such explosion without requiring exclusion of components.
- Maintenance of entailment: When the data or rules used for creating an entailment changes, the set of inferred triples is marked as not 'VALID' and the entailment has to be done again to get it back to a 'VALID' status. Support for limited form of incremental maintenance is currently planned.
- Explanation and Validation: Both are supported.

3.3 Querying RDF and Relational Data

Overview. Querying capabilities in Oracle database semantic store can be characterized as follows:

- Standalone SPARQL support: Oracle supports querying using standalone SPARQL queries via the Jena Interface with underlying Oracle Jena Adaptor implementing the translation to the querying capabilities of Oracle database semantic store.
- SQL-based SPARQL support: This hybrid query language support allows users to specify SPARQL graph-pattern as an argument to a table function, SEM_MATCH, which returns a table of the mappings, each of which is a solution for the specified graph pattern.



Querying RDF data in SPARQL

- Find pairs of siblings (same parents)

-
  ```
  SELECT ?x ?y
  FROM <family> WHERE {
  ?x <hasFather> ?f . ?x <hasMother> ?m .
  ?y <hasFather> ?f . ?y <hasMother> ?m .
  FILTER( ?x != ?y)
  }
  ```

SQL-based RDF Query in Oracle

- Find pairs of siblings (same parents)

-
  ```
  SELECT x, y
  FROM TABLE(SEM_MATCH('{
    ?x <hasFather> ?f . ?x <hasMother> ?m .
    ?y <hasFather> ?f . ?y <hasMother> ?m .
  }',
  sem_models('family'),null,null,null
  ))
  WHERE x != y;
  ```

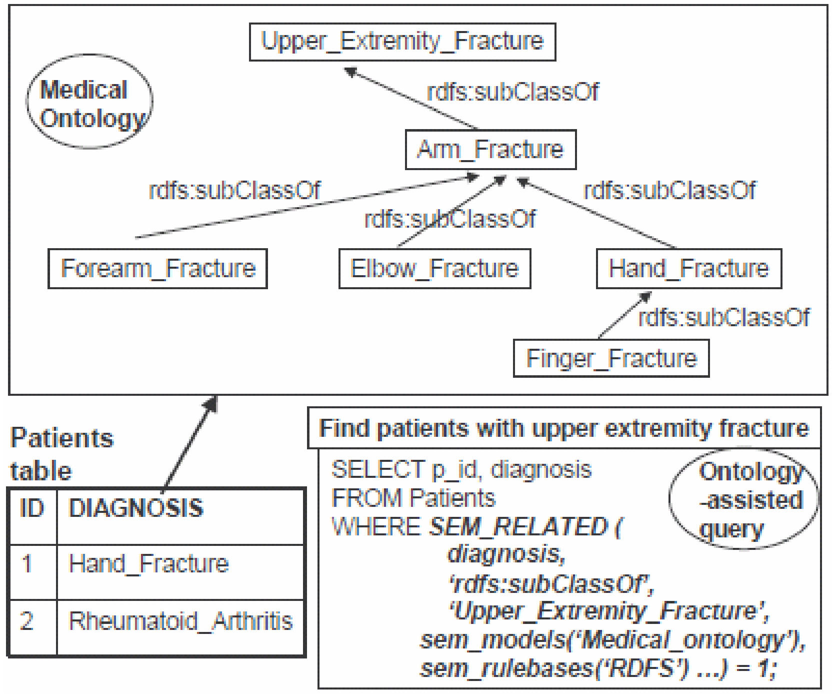
Fig. 4. RDF query for finding pairs of siblings in 'family' RDF model: left side shows the SPARQL query and right side shows the SQL-based RDF query in Oracle

The target dataset could include RDF model(s) and optionally, inferred triples of an entailment (pre-computed for the RDF model(s) and one or more rulebases). The SQL query allows combining the table of mappings returned by SEM_MATCH invocation(s) with relational tables (via joins, for example).

The example in Fig. 4 shows how a SPARQL query for finding siblings in a 'family' RDF model can be expressed as a SQL-based RDF query in Oracle. The SPARQL graph pattern is passed in as the first argument to the SEM_MATCH table function that returns a table of results, each row representing a pair of siblings. As shown in the example, the resulting table can then be further processed by SQL constructs applicable to tables.

- Query rewrite for better performance: The SPARQL query specified in SEM_MATCH is automatically translated to an SQL query and the user-specified SQL query is rewritten using the translation as a substitution for the SEM_MATCH invocation [8]. This allows the resulting single SQL query to be seen as a whole by the optimizer and results in generation of better execution plan. Also, this eliminates the need for communication at run time between table function implementation and the SQL engine.

- Hints for query performance tuning: Users may optionally specify SQL-like hints for suggesting join orders, access methods, etc. to the database optimizer.
- Use of identifiers to avoid joins: In SQL-based querying, users may retrieve the identifiers for the matches for some of the variables instead of the lexical values. This avoids joins with the value↔id mapping table and hence can improve query performance.
- Ontology-assisted querying of relational data: To allow querying of relational tables with filters based on how column values are related to the terms used in an ontology, an operator, SEM_RELATED, is supported in Oracle database semantic store.



ID	DIAGNOSIS
1	Hand_Fracture
2	Rheumatoid_Arthritis

```
SELECT p_id, diagnosis
FROM Patients
WHERE SEM_RELATED (
        diagnosis,
        'rdfs:subClassOf',
        'Upper_Extremity_Fracture',
      sem_models('Medical_ontology'),
      sem_rulebases('RDFS') ...) = 1;
```

Fig. 5. Ontology-assisted querying in Oracle: Querying "Patients" table using the ontology "Medical Ontology" to find patients with "upper extremity fracture"

The SEM_RELATED operator, used in the WHERE clause of a SQL query, takes as input a graph pattern involving use of the column (currently limited to a single triple pattern, specified as a sequence of three arguments, first one being the column, second one an RDF term representing a predicate or its inverse, and the third one an RDF term) and names of the RDF model(s) and optionally any rulebase(s), and returns 0 or 1 depending upon whether a solution is present that matches the graph pattern. In case of a match, ancillary operator SEM_DISTANCE represents the distance between the subject and object in the graph pattern.

Fig. 5 shows one such example. Querying the diagnosis column, in a Patient table, for 'upper extremity fracture' using the traditional '=' relational operator would yield no results, whereas use of the SEM_RELATED operator in the context of the "Medical Ontology" (entailed with RDFS) would return patient id 1 because the value of the diagnosis column, Hand_Fracture, is a subclass of Upper_Extremity_Fracture due to transitivity of the rdfs:subClassOf property in the RDFS entailment regime.

4 Summary

The mature and time tested DBMS technologies can be adopted to store and manage RDF data. RDF does bring interesting requirements in terms of irregular data, lack of schema, or continuously evolving schema. However, with careful database schema design and use of indexes, a scalable and high performance RDF store can be supported in a database. The capability of inferencing, which is an integral part of a RDF store, can also be supported natively in the database.

The case study of implementing RDF store in Oracle Database further validates these aspects. Also, it illustrates the ability to combine semantic (SPARQL graph pattern based) queries over RDF data with traditional SQL queries over enterprise data.

However, there are areas that need further work including support for graph-oriented querying (example, neighborhood of nodes, path between nodes, etc.) over large RDF data sets that do not fit in main memory, inferencing over more expressive OWL profiles, incremental inference, and adoption of traditional report generation tools for RDF to allow rapid development of semantic applications.

Acknowledgments. We acknowledge the contributions of several other people who have contributed extensively in the design and implementation of the Oracle Database semantic technologies namely Nicole Alexander, Melliyal Annamalai, Nipun Bhatia, Eugene Inseok Chong, George Eadon, Vladimir Kolovski, Matthew Perry, Siva Ravada, Seema Sundara, Aravind Yalamanchi, and Zhe Wu. We thank our manager Jay Banerjee, and product managers Xavier Lopez and Bill Beauregard, for their encouragement and support. We also appreciate the feedback from the reviewer.

References

1. Resource Description Framework (RDF), `http://www.w3.org/RDF`
2. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation (February 2004), `http://www.w3.org/TR/rdf-schema`
3. OWL Web Ontology Language Reference, `http://www.w3.org/TR/owl-ref`
4. Pellet: The Open Source OWL DL Reasoner, `http://clarkparsia.com/pellet/`
5. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
6. UniProt, `http://www.uniprot.org/`

7. WordNet, `http://wordnet.princeton.edu/`
8. Chong, E.I., Souripriya Das, S., Eadon, G., Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme. In: VLDB 2005, pp. 1216–1227 (2005)
9. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: Supporting Keyword Columns with Ontology-based Referential Constraints in DBMS. In: ICDE 2006, p. 95 (2006)
10. SPARQL query language for RDF, `http://www.w3.org/TR/rdf-sparql-query`
11. Das, S., Chong, E.I., Wu, Z., Annamalai, M., Srinivasan, J.: A Scalable Scheme for Bulk Loading Large RDF Graphs into Oracle. In: ICDE 2008, pp. 1297–1306 (2008)
12. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In: ICDE 2008, pp. 1239–1248 (2008)
13. Jena – A Semantic Web Framework for Java, `http://jena.sourceforge.net`
14. Oracle Database Semantic Technologies, `http://www.oracle.com/technology/tech/semantic_technologies/`
15. Advances in Virtuoso RDF Triple Storage (Bitmap Indexing), `http://virtuoso.openlinksw.com/wiki/main/Main/VOSBitmapIndexing`
16. Sesame: RDF Schema Querying and Storage, `http://www.openrdf.org/`
17. Semantic Technologies Product Performance, `http://www.oracle.com/technology/tech/semantic_technologies/htdocs/performance.html`
18. Jena Adaptor Release 2.0 for Oracle Database, `http://www.oracle.com/technology/software/tech/semantic_technologies/index.html`
19. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. In: VLDB, pp. 1008–1019 (2008)
20. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB 2007, pp. 411–422 (2007)
21. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Karsten Tolle, K.: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In: SemWeb 2001 (2001)
22. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL basic graph pattern optimization using selectivity estimation. In: WWW 2008, pp. 595–604 (2008)
23. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 685–701. Springer, Heidelberg (2005)
24. Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes Second Edition W3C Recommendation. W3C (October 2004)
25. Virtuoso RDF Views – Getting Started Guide, `http://virtuoso.openlinksw.com/Whitepapers/pdf/Virtuoso_SQL_to_RDF_Mapping.pdf`
26. Bizer, C.: The D2RQ Platform - Treating Non-RDF Relational Databases as Virtual RDF Graphs, `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/`
27. Sahoo, S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr., T., Auer, S., Sequeda, J.: A Survey of Current Approaches for Mapping of Relational Databases to RDF, `http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt`

28. Wilkinson, K., Craig Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: SWDB 2003, pp. 131–150 (2003)
29. Srinivasan, J., Murthy, R., Sundara, S., Agarwal, N., DeFazio, S.: Extensible Indexing: A Framework for Integrating Domain-Specific Indexing into Oracle8i. In: Proceedings of the 16th International Conference on Data Engineering (ICDE), pp. 91–100 (2000)
30. Horst, H.J.: Completeness, Decidability and Complexity of Entailment for RDF Schema and A Semantic Extension Involving the OWL Vocabulary. J. Web Sem. 3(2-3), 79–115 (2005)
31. OWL 2 Web Ontology Language Profiles: W3C Working Draft (April 21, 2009), `http://www.w3.org/TR/2009/WD-owl2-profiles-20090421/`

Technologies for the Social Semantic Desktop

Michael Sintek[1], Siegfried Handschuh[2], Simon Scerri[2], and Ludger van Elst[1]

[1] Knowledge Management Department
German Research Center for Artificial Intelligence (DFKI) GmbH,
Kaiserslautern, Germany
{firstname.surname}@dfki.de

[2] DERI, National University of Ireland, Galway
{firstname.surname}@deri.org

Abstract. The vision of the *Social Semantic Desktop* defines a user's personal information environment as a source and end-point of the *Semantic Web*: Knowledge workers comprehensively express their information and data with respect to their own conceptualizations. *Semantic Web* languages and protocols are used to formalize these conceptualizations and for coordinating local and global information access.

A core challenge is to integrate existing legacy Desktop data into the *Social Semantic Desktop*. Semantic lifting is the process of capturing the semantics of various types of (semi-)structured data and/or non-semantic metadata and translating such data into *Semantic Web* conceptualizations.

From the way the vision of the *Social Semantic Desktop* is being pursued in the NEPOMUK project, we identified several requirements and research questions with respect to knowledge representation. In addition to the general question of the expressivity needed in such a scenario, two main challenges come into focus: i) How can we cope with the heterogeneity of knowledge models and ontologies, esp. multiple knowledge modules with potentially different interpretations? ii) How can we support the tailoring of ontologies towards different needs in various exploiting applications?

In this paper, we present semantic lifting as a means to create semantic metadata and the Nepomuk Representation Language (NRL) as a means to represent these metadata. NRL is an approach to these two aforementioned questions that is based on named graphs for the modularization aspect and a view concept for the tailoring of ontologies. This view concept turned out to be of additional value, as it also provides a mechanism to impose different semantics on the same syntactical structure.

We furthermore present some of the ontologies that have been developed with the help of NRL in the NEPOMUK project to build the semantic foundations for the *Social Semantic Desktop*.

1 Overview

This paper constitutes the material for the lecture on the (Social) Semantic Desktop given at the Reasoning Web Summer School 2009 (http://reasoningweb.org/2009/).

In Sect. 2, we present the basic ideas of the *Social Semantic Desktop*. The remaining sections describe technologies developed and used in NEPOMUK and other projects to build the *Social Semantic Desktop*. Lifting (Sect. 3) is the process of capturing the semantics of various types of (semi-)structured data and/or non-semantic metadata and translating such data into relations, attributes and concepts within an ontology. NRL (Sect. 4) is the NEPOMUK Representational (ontology) Language, developed as an extension to RDF/S with additional support for named graphs and views, in order to fulfill some of the requirements of a representational language for the *Social Semantic Desktop*. Finally, in Sect. 5, we present some of the resulting ontologies that have been developed in NEPOMUK.

2 The Social Semantic Desktop

2.1 Motivation

The very core idea of the *Social Semantic Desktop* is to enable data interoperability on the personal desktop based on *Semantic Web* standards and technologies, *e. g.*, Ontologies and semantic metadata. The vision [13] aims at integrated personal information management as well as at information distribution and collaboration, envisioning two expansion states: i) the *Personal Semantic Desktop* for personal information management and later ii) the *Social Semantic Desktop* for distributed information management and social community aspects.

In traditional desktop architectures, applications are isolated islands of data—each application has its own data, unaware of related and relevant data in other applications. Individual vendors may decide to allow their applications to interoperate, so that, *e. g.*, the email client knows about the address book. However, today there is no consistent approach for allowing interoperation and a system-wide exchange of data between applications. Similarly, the desktops of different users are also isolated islands—there is no standardized architecture for interoperation and data exchange between desktops. Users may exchange data by sending emails or uploading it to a server, but so far there is no means for a seamless communication between an application used by one person on their desktop and an application used by another person on another desktop. The knowledge exchange and integration problem on the desktop is thus similar to that which exists on the Web.

The *Social Semantic Desktop* paradigm adopts ideas from the *Semantic Web* (SW) paradigm [4], which offers a solution for the web. Formal Ontologies capture both a shared conceptualization of desktop data and personal mental models. RDF (Resource Description Format) serves as a common data representation format. Together, these technologies provide a means to build the semantic bridges necessary for data exchange and application integration. The *Social Semantic Desktop* will transform the conventional desktop into a seamless, networked working environment, by loosening the borders between individual applications and the physical workspace of different users. By aligning the

Social Semantic Desktop paradigm with the *Semantic Web* paradigm, a *Semantic Desktop* can be seen as both the source and the end-point of the *Semantic Web*.

2.2 State of the Art

In the following we present a brief review of relevant research and development approaches for the *Social Semantic Desktop*.

Gnowsis [23] was among the first research projects targeting a *Semantic Desktop* system. Its goal was to complement, rather than replace, established desktop applications and the desktop operating system with *Semantic Web* features. The primary focus of Gnowsis was on Personal Information Management (PIM). It also addressed the issues of identification and representation of desktop resources in a unified RDF graph.

The Haystack [20] project presents a good example for an integrated approach to the *Social Semantic Desktop* field. Inter-application barriers are avoided by simply replacing these applications with Haystack's own word processor, email client, image manipulation, instant messaging, *etc.*Haystack allows users to define their own arrangements and connections between views of information, thus making it easier to find information located in the personal space.

The IRIS Semantic Desktop [8] (Integrate. Relate. Infer. Share) provided an application framework that enables users to create a personal map across their office-related information objects.

DeepaMehta [22] is an open source *Semantic Desktop* application based on the Topic Maps standard. The DeepaMehta UI, which runs through a Web browser, renders Topic Maps as a graph, similar to concept maps. Information of any kind as well as relations between information items can be displayed and edited in the same space. The user is no longer confronted with files and programs.

Although the systems we have looked at focused on isolated and complementary aspects, they clearly influenced the vision of the *Social Semantic Desktop* presented in this paper. However our vision is more general and comprehensive.

2.3 Networked Collaborative Knowledge

We all face the problem of having increasingly more information on our desktops. The average workspace covers hundreds of thousands of different files (including emails), some of which we vaguely remember the place in which they were stored. To make matters worse for the desktop user, the web has not only enabled further information creation and dissemination, but has also opened wide the information floodgates. Furthermore, this information is highly confined. The computer desktop is our universal workspace, where we have all kinds of information in different formats, and use it for various purposes in different applications. Some of this data has little explicit representation, is not always suitably structured and is trapped and imprisoned in applications, *i. e.*, Data Silos. We have multiple isolated information spaces on the desktop, *e. g.*, email clients, file systems, music managers, web browsers. The same is true for the collaborative web information system we use, *e. g.*, wikis, sharepoint, BSCW. These data silos prevent us from joint problem solving and collaboration, as well as answering questions whose result is spread across multiple workspaces. In short, they hinder us from exchanging personal content from one workspace to another.

The central idea of the *Social Semantic Desktop* focuses on how social and collaborative activities and their coordination can be improved through semantic technologies. Semantics hold the promise of automatic understanding and better information organization and selective access, and providing standard means for formulating and distributing metadata and Ontologies. Hence, semantic collaborative information management facilitates the integration of information between desktop applications and the Web, *i. e.*, focused and integrated personal information management along with information distribution and collaboration.

Classical collaborative information management takes place in controlled, closed and comparatively small environments. In parallel, the WWW emerged as a phenomenon that is unstructured, social, open, and which distributes information on a large scale. Thus information is often disconnected on the Web. To solve this we require computers to make sense of this information, hence meaning, and thus semantics; to achieve computer-understandable data by exploiting existing resources. These existing resources can be lifted by using formal languages, such as RDF/S or NRL (*cf.* Sect. 4). This enables us to network the data and thus to achieve a higher level of new information.

Although knowledge is inherently strongly interconnected and related to people, this interconnectedness is not reflected or supported by current information infrastructures. The lack of interconnectedness hampers basic information management and problem-solving and collaboration capabilities, like finding, creating and deploying the right knowledge at the right time.

Besides the creation of knowledge through observation, networking of knowledge is the basic process to generate further knowledge. Networking knowledge, can produce a piece of knowledge whose information value is far beyond the mere sum of the individual pieces, *i. e.*, it creates new knowledge. With the Web we now have a foundational infrastructure in place that enables the linking of information on a global scale. Furthermore, with the desktop we have an infrastructure that stores all our personal information models. Adding meaning moves the interlinked information to the knowledge level: Web + Semantics + Desktop = *Social Semantic Desktop*.

Now is the time to tackle the next step: exploiting semantics to create an overall knowledge network that bridges the information islands in order to enable people, organizations and systems to collaborate and interoperate on a global scale.

In the following we will show how a *Social Semantic Desktop* can provide answers to the following questions:

Q1: How do you structure your personal information on your desktop? How do you structure your file system, your email and your bookmarks? Do you use other means to manage your information?

Q2: How do you share and exchange the data with your colleagues? With email—like most people, or with a Wiki, a share point system, *etc.*?

Q3: How do you find an expert in your organization, given it employs many people as to make it hard for you to keep an overview?

2.4 User Mental Models

Representation of the users mental models take the form of a personal information model (*cf.* Sect. 5.3). Lets envision an average desktop user called Claudia (*cf.* Fig. 1), who is organizing her information in folders and emails. A close look at it will reveal common topics in both structures, such as projects, organization, people, topics, *etc.*These mental models are currently isolated in her applications, and the goal of the *Semantic Desktop* is to free this information and represent it explicitly.

Therefore we propose to apply *Semantic Web* technologies to represent these mental models, by utilizing existing and/or extended standards and RDF/S vocabularies such as VCard for an optimal information representation and then to lift (*cf.* Sect.3) the existing structured data up to a NRL representation (*cf.* Sect. 4); thereby allowing the structuring of the mental model only once and not several times.

2.5 Interconnected Desktops

The explicit classification scheme, encapsulated within the Personal Information Model PIMO (*cf.* Sect. 5.3), helps individuals manage their desktop information. The semantics of NRL allows for the automatic processing of this model and the deduction of new knowledge. It creates a kind of a personal semantic web: a semantically-extended supplement to the user's view of their personal information.

This research contributes to the so-called vocabulary "onion" by providing PIMO, NIE (*cf.* Sect. 5.2) and other required vocabularies. An instance of the PIMO represents a single user's concepts, such as projects, tasks, organizations, *etc.*The NIE set of ontologies provides vocabularies for describing information elements (such as files, contact, calendar data, emails-s) which are commonly present on the desktop and in collaborative systems.

By interconnecting such personal semantic desktops (*cf.* Fig. 1), *e. g.*, via a client server model or the use of P2P technology; we can easily exchange information. Not only can we exchange data in the form of documents, but also structured information about project, people, events, *etc.*; as well as the personal models themselves. For example, Claudia might have developed a very good structure for a project in which Dirk is also working, so she shares this structure with Dirk, hence allowing him to re-use this information. Note that this is not possible with current desktop systems—one cannot easily transfer their project file folder structure to a colleague.

On top of the P2P networking (which allows content-based routing) we have social protocols and algorithms that enable an explicit representation of relationships which is similar to social systems (*e. g.*, LinkedIn[1] and others), yet is

[1] http://www.linkedin.com/

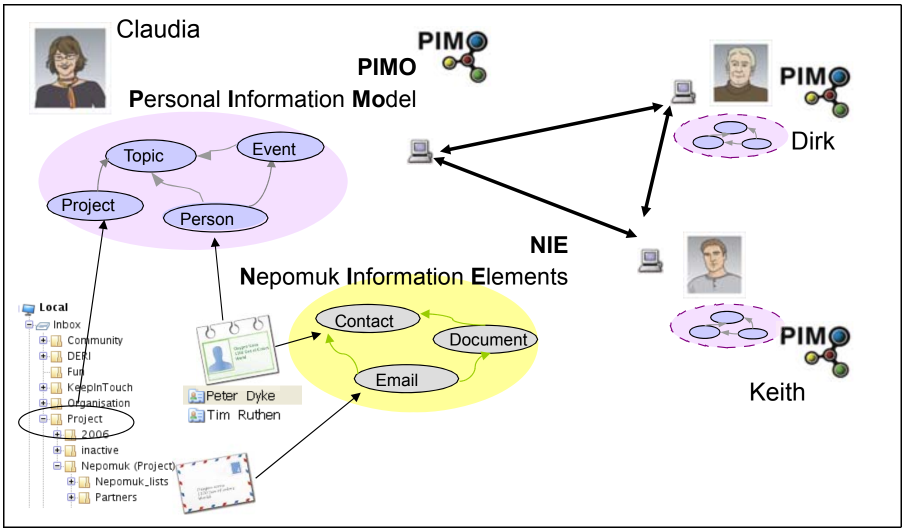


Fig. 1. Interconnected *Social Semantic Desktop*s

open in the sense that it allows for the creation of new connections and the establishment of new relationships. This accelerates collaboration and allows for the maintenance of shared views.

2.6 Achievements

We can conclude that via the *Social Semantic Desktop* we achieve a universal platform for:

– Personal Information Management
– Distributed Information Management
– Social Expansion and Community Creation

The impact results in dramatic time savings, by i) filtering out marginal information, ii) discovering vital information and building, as well as participating, in communities of practice.

We manage personal information by mapping native structures onto our own mental models and representing data in a unified way. The social aspect of sharing and community building in an organization is done by connecting individual semantic desktops.

The answers to the previous questions, with the *Social Semantic Desktop* now in the picture, are thus:

A1: The user can manage and structure their personal information (mental model) via PIMO.

A2: The user can share and exchange their personal information via the *Social Semantic Desktop* network, which allows for a content-based routing and a "link routing" based on social connections.

A3: The user can find experts within their social circle by using intelligent services on top of the *Social Semantic Desktop* infrastructure. These utilize the interest profile of the users' PIMOs to detect and classify experts and communities.

In the following chapters we will learn about the foundational technology which enables the realization of the here presented general issues, *i. e.*, methodologies for the lifting of existing data onto personal information models, and the semantic backbone of the *Social Semantic Desktop*—consisting of NRL and the rest of the NEPOMUK Ontologies.

3 Semantic Lifting and Human Language Technologies for the Semantic Desktop

3.1 Background

The *Social Semantic Desktop* requires metadata represented in RDF/NRL (*cf.* Sect. 4) to operate. The RDF metadata can be the result of the following processes:

– i) Lifting of existing structured data onto RDF
– ii) Usage of Human Language Technology (HLT) to capture knowledge from text and transform that into RDF
– ii) Manual creation of metadata by linking, annotation or tagging

In this chapter we will focus on lifting and HLT. Semantic lifting is the process of capturing the semantics of various types of (semi-)structured data and/or non-semantic metadata and translating such data into relations, attributes and concepts within an ontology. Candidate data for lifting includes non-semantic metadata (*e. g.*, in XML), emails, directory structures, files on disk, IMAP mailboxes, address books and schemas such as iCalendar[2]. The core challenges are to integrate existing legacy Desktop data into the *Social Semantic Desktop* (*cf.* Fig. 2); to expose or make explicit such data to both the *Social Semantic Desktop* and the *Semantic Web*; to reuse, rather than replace, existing data; and to enhance, rather then replace, existing applications.

Human language technology (HLT), in its broadest sense, can be described as computational methods for processing and manipulating language, for instance text analysis, information extraction or controlled language. This technology has materialized on the *Semantic Desktop* in the form of integration into the user's email client, personal meeting note-taker as well an automated textual analysis of documents on their desktop.

[2] `http://en.wikipedia.org/wiki/ICalendar`

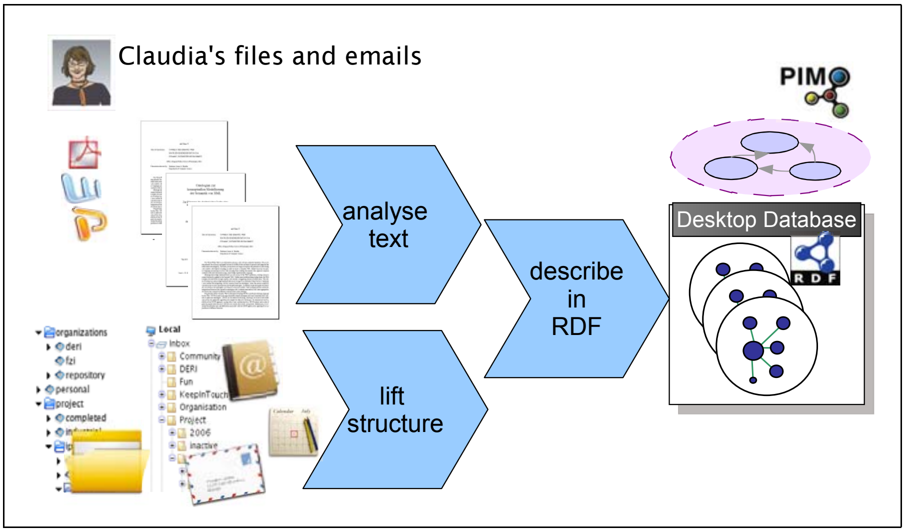


Fig. 2. Lifting unstructured data onto standard semantic representations

3.2 Lifting on the Semantic Desktop

In most cases, the process of lifting structured information onto an RDF layer indexes data that exists in desktop applications. The data is converted to standard vocabularies and stored in an RDF repository which serves as a local storage. Existing systems have implemented a lifting service for a *Semantic Desktop* up to varying degrees, *e. g.*, Aperture, which is a Java application; Beagle++, a Linux-Gnome desktop crawler; and Strigi, which is part of KDE 4. We will now have a closer look at these systems.

Aperture: Aperture[3] is a cross-platform java project. Aperture extracts data from various information sources by crawling each source. It transforms the data from the existing formats to RDF, using a set of purposely-developed Ontologies. Aperture only does the crawling and extraction, since storage is usually handled by a Sesame[4] RDF data store.

Beagle++: Beagle++[5] is based on Gnome Beagle. Beagle is a desktop and search application. Gnome Beagle consists of sets of Backends and Filters. Every backend is in charge of extracting metadata from various data sources.

Semantic extensions for extractors of Gnome Beagle towards Beagle++ are: i) Path Annotation with WordNet, ii) Web Cache Metadata Generation, and iii) Publication Metadata Generation from PDF files.

3 http://aperture.sourceforge.net/
4 http://www.openrdf.org/
5 http://beagle2.kbs.uni-hannover.de/

Strigi/Soprano: Nepomuk-KDE[6] uses Strigi[7] and Soprano[8] as core component for data lifting. Strigi, in a similar fashion to Aperture and Beagle++, crawls the data available on the hard disk and extracts the file metadata as well as the content of the files (where it makes sense to do so). As an example, audio files often carry information about the artist in their metadata. On the other hand while PDF files can contain metadata about the author, the author can also be referred to in the content of the PDF file itself. Soprano runs in the Nepomuk storage process. Strigi reads the data out of the files and passes the information into Nepomuk/Soprano. Sesame[9] or Redland[10] are RDF repository backends for Soprano. Soprano fully supports both PIMO and NIE as valid data formats.

3.3 Human Language Technology on the Semantic Desktop

In this section we brief the application of Human Language Technology (HLT) to extract information from textual documents, and how techniques like controlled language and natural language generation can be utilized to generate user-friendly interfaces to the *Semantic Desktop*.

HLT applied on Textual Content. We first have a look at the application of HLT for information extraction from textual content in order to create NRL-based metadata from documents.

Keyphrase Extraction On the *Semantic Desktop* keyphrases are an important instrument for cataloging and information retrieval purposes, *e. g.*, Keyphrases can be used for Semantic Tagging. In literature research, they provide a high-level and concise summary of the content of textual documents or the topics covered therein, allowing humans to quickly decide whether a given text is relevant. As the amount of textual content on desktops grows fast, keyphrases can contribute to manage large amounts of textual information, for instance by marking up important sections in documents, *i. e.*, to provide increased user experience in document exploration.

As keyphrases are a description of textual data, the consideration of HLT tools in order to automate the extraction process is obvious. While shallow techniques are a long way from language understanding, in combination with statistical processing they can be helpful in many ways, providing a first stop in automatic content-metadata extraction, which then can be used as input for more sophisticated technologies.

The main idea here is to use the keyphrases as a first step to propose a Semantic Tag in order to annotate a document (*cf.* Sect. 5.1). The reduced set of keyphrase candidates will provide a less noisy summary of the topics mentioned in a document. This reduced set, in fact, does enable querying ontology

[6] http://nepomuk.kde.org/
[7] http://strigi.sourceforge.net/
[8] http://soprano.sourceforge.net/
[9] http://www.openrdf.org/
[10] http://librdf.org/

libraries (*e. g.*, OntoSelect[11] or Watson[12]) for good-fitting schemes, which then can be retrieved for further semantic annotation in addition to the Semantic Tags provided by the keyphrases.

Speech Act Detection One of the applied HLT on text technology of the *Semantic Desktop* is based on speech act detection in email and instant messaging conversations. The notion of a speech act pursued here is based on that defined by John Searle [31]. At its most basic definition a speech act is an utterance, understood more specifically as a performative utterance or an illocutionary act (a term introduced by John L. Austin [1]), where it is assumed that by saying something one actually is doing something. In our case, the utterances take the form of typed text. For instance, a sentence from Claudia's subordinate asking her politely to attend an important meeting, expresses the speaker's (or the sender of the sentence) wish for Claudia to attend, and sets a new requirement for Claudia—to reply to the meeting suggestion. On the contrary, the same sentence from Claudia's manager will also express the wish of the sender for Claudia to attend, but the expected requirement for Claudia will be to attend the meeting without further ado.

Semanta[13] is a fully-implemented system supporting Semantic Email, whereby we have lifted email processes to a semantic level via speech act theory and a formally-defined ad-hoc email workflow model. In our approach we considered the fact that an email has one or more purposes, or Action Items. The content of an email message can be summarized into a number of such items (*e. g.*, Meeting Request, Task Assignment, File Request, *etc.*). Once exchanged, every single action item can be seen as the start, or continuation of a separate workflow.

The sMail Conceptual Framework [28] applies Speech Act Theory [31] to the email communication process, in order to provide a formal structure and semantics for these action items and their workflows. Email action items like the ones above can be represented by a number of speech act instances provided in the sMail ontology[14]. The Email Speech Act Workflow model [27] is then used to support the user with handling email workflows, *e. g.*, providing them with a set of options when reacting to action items in email.

Computational linguistics technologies, namely Ontology-Based Information Extraction (OBIE) techniques, are employed to provide semiautomatic annotation of action items (speech acts) in email content. The information extraction is based on a declarative model which classifies text into speech acts based on a number of linguistic features like sentence form, tense, modality and the semantic roles of verbs. The system deploys a GATE [9] corpus pipeline consisting of a tokenizer, modified sentence splitter, POS tagger, keyphrase lookup via Finite State gazetteers and several JAPE [10] grammars. Email annotations are represented in RDF, using a number of Ontologies (sMail, NMO, NCO, *etc.*) and embedded within email messages. This enables semantic email to be used

[11] olp.dfki.de/ontoselect/
[12] http://watson.kmi.open.ac.uk/WatsonWUI/
[13] http://smile.deri.ie/projects/semanta
[14] http://ontologies.smile.deri.ie/smail

as a vessel for the transportation and also the sharing of semantics across social semantic desktops.

HLT to generate Interfaces. HLT can be applied to Controlled Language, Natural Language Generation and Document Analysis in order to provide a user-friendly interface to the *Semantic Desktop*. Below we provide examples of how these techniques were utilised.

Controlled Language Interfaces Our research investigates how HL) Interfaces, specifically Controlled Natural Languages (CNL) and applied Natural Language Generation(NLG) can provide a user-friendly means for the non-expert users or small organizations to exploit *Semantic Web* technologies specifically on the *Social Semantic Desktop*.

Roundtrip Ontology Authoring[12] (ROA) is a process that allows non-expert users to author or amend an ontology by using simple, easy-to-learn, controlled natural language. The process is a combination of Controlled Language for Information Extraction (CLIE) and Text Generation which is developed on top of GATE.

Furthermore Controlled Language (CNL) [11] offers an incentive to the novice user to annotate, while simultaneously authoring his/her respective documents in a user-friendly manner, but simultaneously shielding him/her from the underlying complex knowledge representation formalisms. A natural overlap exists between tools, used for both ontology creation and semantic annotation. However, there is a subtle difference between both processes. Semantic annotation has been described as both a process, as well as the outcome of the process. Hence it describes i) the process of addition of semantic data or metadata to the content given an agreed ontology and ii) the semantic data or metadata itself as a result of this process. Of particular importance here is the notion of the addition or association of semantic data or metadata to content.

Personalized Visual Document Collection Analysis The PIMO ontology can also be used to aid scientists and analysts alike in exploring a text collection in a personalized manner in addition to being a formal representation of parts of knowledge workers' spheres of interest.

Apart from the need to retrieve information from documents that are relevant to certain topics of interest, knowledge workers often also need to explore and analyze a collection of documents as a whole, to gain further understanding. Unlike the information retrieval activity, the information analysis activity aims to provide the users with an overall picture of a text collection as a whole, on various dimensions instead of presenting them with the most relevant documents satisfying some search criteria. Given the amount and the unstructured or weakly-structured nature of textual documents that analysts have to deal with, developments in visualization research are beneficial in helping them to gain needed insights in a timely manner.

In this context, we utilized an innovative visualization approach, called IVEA [35,36], which leverages upon the PIMO ontology and the Coordinated Multiple Views technique to support the personalized exploration and analysis of document collections. IVEA allows for an interactive and user-controlled exploration process in which the knowledge workers can gain meaningful, rapid understanding about a text collection via intuitive visual displays. Not only does it allow the users to integrate their interests into the visual exploration and analysis activity, but it also enables them to incrementally enrich their PIMO Ontologies with entities matching their evolving interests in the process. With the newly added entities, the PIMO ontology becomes a richer and better representation of the users' interests and hence can lead to better and more personalized exploration and analysis experiences in the future. Furthermore, not only can IVEA be beneficial to its targeted task, but it also provides an easy and incremental way that requires minimal effort from the users to keep their PIMO Ontologies in line with their continuously changing interests. This, indirectly, can also benefit other PIMO-based applications. The work leverages upon research in information visualization, information retrieval, and human language technology for semantic annotation. These technologies are used in support of the larger HCI goal of enabling effective personalized interactions between users and text collections.

In the next chapters, we will learn about the underlying representation formalism for the lifted knowledge—NRL and the other Nepomuk Ontologies.

4 NRL—The NEPOMUK Representational Language

4.1 Motivation

The viewpoint of the user comprehensively generating, manipulating and exploiting private as well as shared and public data has to be adequately reflected in the representational basis of a *Social Semantic Desktop*. While we think in general the assumptions of knowledge representation in the *Semantic Web* are a good starting point, the *Semantic Desktop* scenario generates special requirements. We identified two core questions which we try to tackle in the knowledge representation approach presented in this paper:

1. How can we cope with the heterogeneity of knowledge models and ontologies, esp. multiple knowledge modules with potentially different interpretation schemes?
2. How can we support the tailoring of ontologies towards different needs in various exploiting applications?

The first question is rooted in the fact that with heterogeneous generation and exploitation of knowledge there is no "master instance" which defines and ensures the "interpretation sovereignty." The second question turned out to be an important prerequisite for a clean ontology design on the semantic desktop, as many applications shall use a knowledge worker's "personal ontology."

From these general questions, we outlined the following five main requirements for knowledge representation on the *Social Semantic Desktop*:

Epistemological adequacy of modeling primitives: In the *Social Semantic Desktop* scenario, knowledge modeling is not only performed offline (*e. g.*, by a distinguished knowledge engineer), but also by the end user, much like in the tagging systems of the Web 2.0 where a user can continuously invent new vocabulary for describing his information items. Even if much of the complexity of the underlying representation formalism can be hidden by adequate user interfaces, it is desirable that there is no big *epistemological gap* between the way an end-user would like to express his knowledge and the way it is represented in the system.

Integration of open-world and closed-world assumptions: The main principle of the SW is that it is an open world in which documents can add new information about existing resources. Since the Web is a huge place in which everything can link to anything else, it is impossible to rule out that a statement could be true, or could become true in the future. Hence, the global semantic web relies on a open-world semantic, with no unique-name assumption—the official OWL and RDF/S semantics. On the other hand, the main principle on the personal *Semantic Desktop* is that it is a closed-world as it mainly focuses on personal data. While most people find it difficult to understand the logical meaning and potential inferences statements of the open-world assumption, the closed-world assumption is easier to understand for the user. Hence, the Personal Semantic Desktop requires the closed-world semantics with a unique-name assumption or good smushing techniques to achieve the same effects. The next stage of expansion of the personal semantic desktop is the *Social Semantic Desktop*, which connects the individual desktops. This will require open-world semantics (in between desktops) with local closed-world semantics (on the personal desktop). Thus the desktop needs to be able to handle external data with open-world semantics. Therefore we require a scenario where we can always distinguish between data per se and the semantics or assumptions on that data. If these are handled analogously, the semantic desktop, a closed-world in theory, will also be able to handle data with open-world semantics.

Handling of multiple models: In order to adequately represent the social dimension of distributed knowledge generation and usage [37], a module concept is desirable which supports encapsulation of statements and the possibility to refer to such modules. The social aspect requires a support for provenance and trust information, when it comes to importing and exporting data. With the present RDF model, importing external RDF data from another desktop presents some difficulties, mainly revolving around the fact that there are no standard means of retaining provenance information of imported data. This means that data is propagated over multiple desktops, with no information regarding the original provider and other crucial information like the context under which that data is valid. This can result in various situations like ending up with outdated RDF data with no means to update it, as well as redundant RDF data which cannot be entirely and safely removed.

Multiple semantics: As stated before, the aspect of distributed (and independently created) information requires the support of the open-world assumption (as we have it in OWL and RDF/S), whereas local information created on a single desktop will have closed-world semantics. Therefore, applications will be forced to deal with different kinds of semantics.

Multiple views: Also required by the social aspect is the support for multiple views, since different individuals on different desktops might be interested in different aspects of the data. A view is dynamic, virtual data computed or collated from the original data. The best view for a particular purpose depends on the information the user needs.

In the next section, we will briefly discuss the state of the art which served as input for the NEPOMUK Representation Language (NRL, [33,34])[15]. Sec. 4.3 gives an overview of our approach. The following sections elaborate on two important aspects of NRL, the *Named Graphs* for handling multiple models (Sec. 4.4) and the *Graph Views* for imposing different semantics on and application-oriented tailoring of models (Sec. 4.5). In Sec. 4.6, we present an example which shows how the concepts presented in this paper can be applied. Sec. 6 summarizes the NRL approach and discusses next steps.

4.2 State of the Art

The Resource Description Framework [17] and the associated schema language RDFS [5] set a standard for the *Semantic Web*, providing a representational language whereby resources on the web can be mapped to designated classes of objects in some shared knowledge domain, and subsequently described and related through applicable object properties. With the gradual acceptance of the *Semantic Web* as an achievable rather than just an ideal World Wide Web scenario, and adoption of RDF/S as the standard for describing and manipulating semantic web data, there have been many attempts to improve some RDF/S shortcomings to handling such data. Most where in the form of representational languages that extend RDF/S, the most notable of which is OWL [2]. Other work attempted to provide further functionalities on top of semantic data to that provided by RDF/S by revising the RDF model itself. The most successful idea perhaps is the named graph paradigm, where identifying multiple RDF graphs and naming them with distinct URIs is believed to provide useful additional functionality on top of the RDF model. Given that named graphs are manageable sets of data in an otherwise structureless RDF triple space composed of all existent RDF data, most of the practical problems arising from dealing with RDF data, like dealing with invalid or outdated data as well as issues of provenance and trust, could be addressed more easily if the RDF model supports named graphs. The RDF recommendation itself does not provide suitable mechanisms for talking about graphs or define relations between graphs [3,17,5,14]. Although the extension of the RDF model with named graph support has been proposed [7,32,19], and the motivation and ideas are clearly stated, a concrete extension to the RDF model supporting named graph has not yet materialized. So far, a basic syntax and semantics that models minimal manipulation of named graphs

[15] Full specifications available at `http://www.semanticdesktop.org/ontologies/nrl/`

has been presented by participants of the Semantic Web Interest Group.[16] Their intent is to introduce the technology to the W3C process once initial versions are finalized. The SPARQL query language [19], currently undergoing standardization by the W3C, is the most successful attempt to provide a standard query language for RDF data. SPARQL's full support for named graphs has encouraged further research in the area. The concept of modularized RDF knowledge bases (in the spirit of named graphs) plus views that can be used to realize the semantics of a module (with the help of rules), amongst other things, has been introduced in the *Semantic Web* rule language TRIPLE [32]. Recently, [30] introduced the concept of Networked Graphs, which are a declarative mechanism to define views over distributed RDF graphs with the help of SPARQL rules.

Since the existing approaches are incomplete wrt. the needs of NEPOMUK and most *Semantic Web* scenarios in general, we propose a combination of named graphs and TRIPLE's view concept as the basis for NRL, the representational language we are presenting. In contrast to TRIPLE, we will add the ability to define views as an extension of RDF and named graphs at the ontological level, thus we are not dependent on a specific rule formalism as in the case of TRIPLE.

In the rest of the NRL section, we will give a detailed description of the named graphs and views features of NRL. Other features of NRL (which consist of some RDFS extensions mainly inspired by Protégé and OWL) will not be discussed.

4.3 Knowledge Representation on the Social Semantic Desktop: The NRL Approach

NRL was inspired by the need for a robust representational language for the *Social Semantic Desktop*, that targets the shortcomings of RDF/S. NRL was designed to fulfill requirements for the NEPOMUK *Social Semantic Desktop* project,[17] hence the particular naming, but it is otherwise domain-independent.

As discussed in the previous section, the most notable shortcoming of the RDF model is the lack of support for handling multiple models. In theory Named Graphs solve this problem since they are identifiable, modularized sets of data. Through this intermediate layer handling RDF data, *e. g.*, exchanging data and keeping track of data provenance information, is much more manageable. This has a great influence in the social aspect of the *Social Semantic Desktop* project, since the success of this particular aspect depends largely on how to successfully deal with these issues. All data handling on the semantic desktop including storage, retrieval and exchange, will therefore be carried out through RDF graphs. Alongside provenance data, more useful information can be attached to named graphs. In particular we feel that named graphs should be distinguished by their roles, *e. g.*, Ontology or Instance Base.

Desktop users may be interested in different aspects of data in a named graph at different times. Looking at the contents of an image folder for instance, the user might wish to see related concepts for an image, or any other files related to

[16] `http://www.w3.org/2004/03/trix/`
[17] `http://nepomuk.semanticdesktop.org/`

it, but not necessarily both concurrently even if the information is stored in the same graph. Additionally, advanced users might require to see data that is not usually visible to regular users, like additional indirect concepts related to the file. This would require the viewing application to realize the RDF/S semantics over the data to yield more results. The desktop system is therefore required to work with extended or restricted versions of named graphs in different situations. However, we believe that such manipulations over named graphs should not have a permanent impact on the data in question. Conversely, we believe that the original named graph should be independent of any kind of workable interpretation executed by an application, which can be discarded if and when they are no longer needed.

For this reason, we present the concept of Graph Views as one of the core concepts in NRL. By allowing for arbitrary tailored interpretations for any established named graph, graph views fulfill our idea that named graphs should not innately carry any realized semantics or assumptions, unless they are themselves views on other graphs for exactly that purpose, and that they should remain unchanged and independent of any view applied on them. This means that different semantics can be realized for different graphs if required. In practice, different application on the semantic desktop will require to apply different semantics, or assumptions on semantics, to named graphs. In this way, although the semantic desktop operates in a closed-world, it is also possible to work with open-world semantic views over a graph. Importing a named graph with predefined open-world semantics on the semantic desktop is therefore possible. If required (and

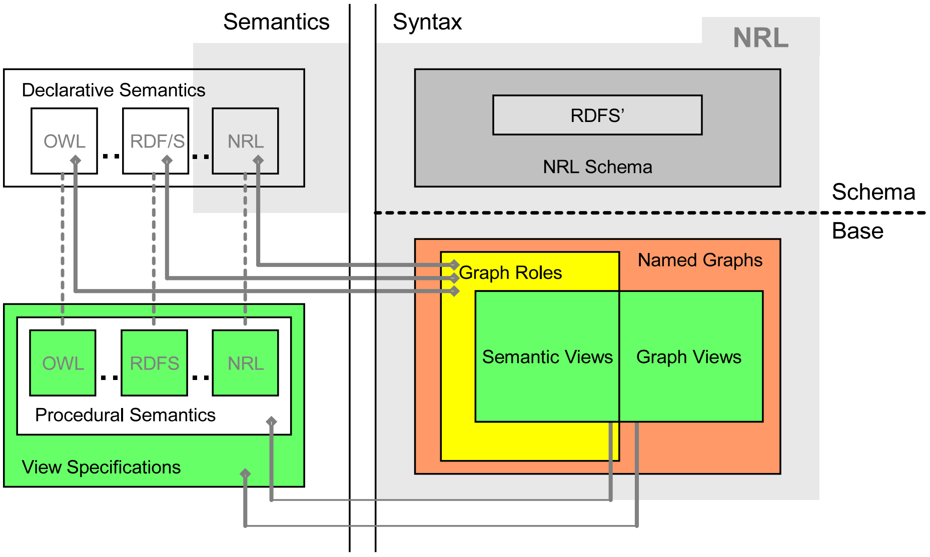


Fig. 3. Overview of NRL—Abstract Syntax, Concepts and Semantics

meaningful), closed-world applications can then work with a closed-world semantics view over the imported graph.

Fig. 3 gives an overview of the components of NRL, depicting both the syntactical and the semantic blocks of NRL. The syntax box contains, in the upper part, the NRL Schema language, which is mainly an extension of (a large subset of) RDFS. The lower part shows how named graphs, graph roles, and views are related, which will be explained in detail in the rest of this paper.

The left half of the figure sheds some light on the semantics of NRL, which has a declarative and a procedural part. Declarative semantics is linked with graph roles, *i. e.*, roles are used to assign meaning to named graphs (note that not all named graphs or views must be assigned some declarative semantics, *e. g.*, in cases when the semantics is (not) yet known or simply not relevant). Views are also linked to view specifications, which function as a mechanism to express procedural semantics, *e. g.*, by using a rule system. The procedural semantics has, of course, to realize the declarative semantics that is assigned to a semantic view.

4.4 Handling Multiple Models: NRL Named Graphs

Named graphs (NGs) are an extension on top of RDF, where every distinct RDF graph is identified by a unique name. NGs provide additional functionality on top of RDF particularly with respect to metametadata (metadata about metadata), provenance, and data (in)equivalence issues, besides making data handling more manageable. Our approach is based on the work described in [7] excluding however, the open-world assumption stated there. As stated earlier (*cf.* Sec. 4.3) we believe that named graphs should not innately carry any realized semantics or assumptions on the semantics. Therefore, despite being designed as a requirement for the Semantic Desktop, which operates under a closed-world scenario, NRL itself does not impose closed-world semantics on data. This and other semantics can instead be realized through designated views on graphs.

A named graph is a pair (n, g), where n is a unique URI reference denoting the assigned name for the graph g. Such a mapping fixes the graph g corresponding to n in a rigid, non-extensible way. The URI representing n can then be used from any location to refer to the corresponding set of triples belonging to the graph g. A graph g' consistent[18] with a distinct graph g named n cannot be assigned the same name n.

An RDF triple can exist in a named graph or outside any named graph. However, for consistency reasons, all triples must be assigned to some named graph. For this reason NRL provides a special named graph, `nrl:DefaultGraph`. Triples existing outside any named graph are considered part of this default graph. This ensures backward compatibility with triples that are not based on named graphs. This approach gives rise to the term RDF Dataset as defined in [19]. An RDF dataset is composed of a default graph and a finite number of distinct named

[18] Two different datasets asserting two unique graphs but having the same URI for a name contradict one another.

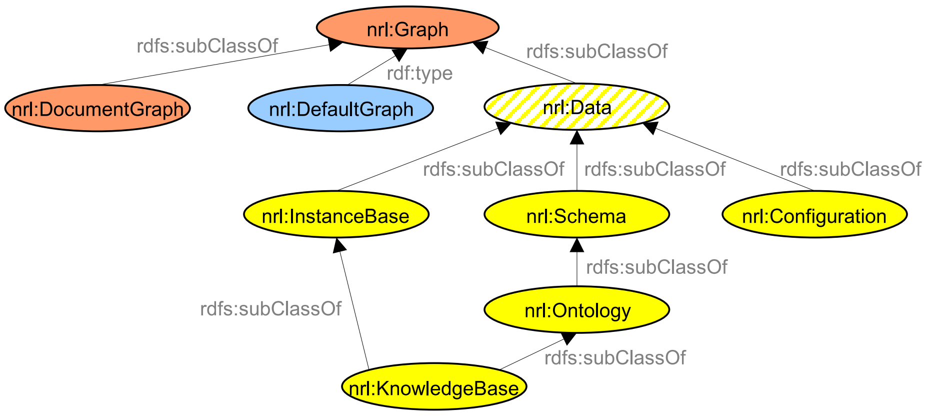


Fig. 4. NRL Named Graph Class Hierarchy

graph, formally defined as the set $\{g, (n_1, g_1), (n_2, g_2), ..., (n_n, g_n)\}$ comprising of the default graph g and zero or more named graphs (n_i, g_i).

NRL distinguishes between graphs and graph roles, in order to have orthogonal modeling primitives for defining graphs and for specifying their role. A graph role refers to the characteristics and content of a named graph (*e. g.*, simple data, an ontology, a knowledge base, *etc.*) and how the data is intended to be handled. NRL provides basic Graph Metadata Vocabulary for annotating graph roles, which vocabulary is extended in the Nepomuk Annotation Ontology (NAO)[19]. Graph metadata is attached to roles rather than to the graphs themselves, because its more intuitive to annotate an ontology, for example, rather than the underlying graph. Roles are more stable than the graphs they represent, and while the graph for a particular role might change constantly, evolution of the role itself is less frequent. An instantiation of a role represents specific type of graph and the corresponding triple set data.

Fig. 4 depicts the class hierarchy supporting NGs in NRL. Graph roles are defined as specialization of the general graph representation `nrl:Data`. A special graph, `nrl:DocumentGraph`, is used as a marker class for graphs that are represented within and identified by a document URL. We now present the NRL vocabulary supporting named graphs. General graph vocabulary is defined in Sec. 4.4 while Sec. 4.4 is dedicated entirely to graph roles.

Graph Core Vocabulary

nrl:Graph and nrl:DocumentGraph. Instances of these classes represent named graphs. The name of the instance coincides with the name of the graph. The graph content for a `nrl:DocumentGraph` is located at the URL that is the URIref for the `nrl:DocumentGraph` instance. This allows existing

[19] `http://www.semanticdesktop.org/ontologies/2007/08/15/nao`

RDF files to be re-used as named graphs, avoiding the need of a syntax like TriG[20] to define named graphs.

nrl:subGraphOf, nrl:superGraphOf, and nrl:equivalentGraph. These relations between named graphs have the obvious semantics: they are defined as $\subseteq$, $\supseteq$, and $=$ on the bare triple sets in these graphs.

nrl:imports is a subproperty of `nrl:superGraphOf` and models graph imports. Apart from implying the $\supseteq$ relation between the triple sets, it also requires that the semantics of the two graphs is compatible if used on, *e. g.*, graphs that are ontologies.

nrl:DefaultGraph. This instance of `nrl:Graph` represents the graph containing all triples existing outside any user-defined named graph. Since we do not apply any semantics to triples automatically, this allows views to be defined on top of triples defined outside of all named graphs analogously to the named-graph case.

Graph Roles Vocabulary

nrl:Data. This subclass of `nrl:Graph` is an abstract class to make graph roles easy-to-use marker classes. It represents the most generic role that a graph can have, namely that it contains data.

nrl:Schema and nrl:Ontology are roles for graphs that represent data in some kind of conceptualization model. `nrl:Ontology` is a subclass of `nrl:Schema`.

nrl:InstanceBase marks a named graph to contain instances from schemas or ontologies. The properties `nrl:hasSchema` and `nrl:hasOntology` relate an instance base to the corresponding schema or ontology.

nrl:KnowledgeBase marks a named graph as containing a conceptual model plus instances from schemas or ontologies.

nrl:GraphMetadata is used to mark graphs whose sole purpose is to store metadata about other graphs. Data about a graph (Graph Metadata) is thus stored in a corresponding graph having this role. The property `nrl:graphMetadataFor` binds a metadata graph to the graph being annotated. Although a graph can have multiple metadata graphs describing it, there can only be one unique metadata graph which defines the graph's important core properties, e.g. whether it is updatable (through `nrl:updatable`) or otherwise. NRL provides the `nrl:coreGraphMetadataFor` property for this purpose, as a subproperty of `nrl:graphMetadataFor`, to identify the core metadata graph for a graph.

nrl:Configuration is used to represent technical configuration data that is irrelevant to general semantic web data within a graph. Other additional roles serving different purposes might be added in the future.

nrl:Semantics. Declarative semantics for a graph role can be specified by referring to instances of this class via `nrl:hasSemantics`. These will usually link (via `nrl:semanticsDefinedBy`) to a document specifying the semantics in a human readable or formal way (*e. g.*, the RDF Semantics document [14]).

[20] `http://sites.wiwiss.fu-berlin.de/suhl/bizer/TriG/`

4.5 Imposing Semantics on Graphs: NRL Graph Views

A named graph consists only of the enumerated triples in the triple set associated with the name, and does not inherently carry any form of semantics (apart from the basic RDF semantics). However in many situations it is desirable to work with an extended or restricted interpretation of simple syntax-only named graphs. These can be realized by applying some algorithm (*e. g.*, specified through rules) which enhances named graphs with entailment triples, returns a restricted form of the triple set, or an entirely new triple set. To preserve the integrity of a named graph, interpretations of one named graph should never replace the original. To model this functionality and retain the separation between original named graph and any number of their interpretations, we introduce the concept of *Graph Views*.

Views are different interpretations for a particular named graph. Formally, a view is an executable specification of an input graph into a corresponding output graph. Informally, they can be seen as arbitrary wrappings for a named graph. Fig. 5 depicts graph view support in NRL. Views are themselves named graphs. Therefore one can have a named graph that is a different interpretation, or view, of another named graph. This modeling can be applied recurrently, yielding a view of a view and so on.

View specifications can execute the view realization for a view, via a set of queries/rules in a query/rule language (*e. g.*, a SPARQL query over a named graph[21]), or via an external application (*e. g.*, an application that returns the transitive closure of `rdfs:subClassOf`). As in the latter example, view realizations can also realize the implicit semantics of a graph according to some language or schema (*e. g.*, RDFS, OWL, NRL *etc.*). We refer to these as *Semantic Views*, represented in Fig. 5 by the intersection of `nrl:GraphView` and graph roles. One can draw a parallel between this figure and Fig. 3. In contrast to graph roles, which have only declarative semantics defined through the `nrl:hasSemantics` property, semantic views also carry procedural semantics, since the semantics of these graphs are always realized, (through `nrl:realizes`) and not simply implied.

Views Vocabulary. In this section we briefly present the NRL vocabulary supporting graph view specifications.

nrl:GraphView represents a view, modeled as a subclass of named graph. A view is realized through a view specification, defined by an instance of `nrl:ViewSpecification` via `nrl:hasSpecification`. The named graph on

[21] A way for using SPARQL to realize view definitions (called Networked Graphs) has been described in [30]. While Networked Graphs allow views to be defined in a declarative way (in contrast to NRL's somewhat procedural way), they lack many of the features we think are important for a view language, *e. g.*, they do do not allow access to the underlying RDF graphs without any interpretation, and they only allow views to be defined via SPARQL which excludes languages with more advanced semantics like OWL and also languages that do not have a declarative semantics.

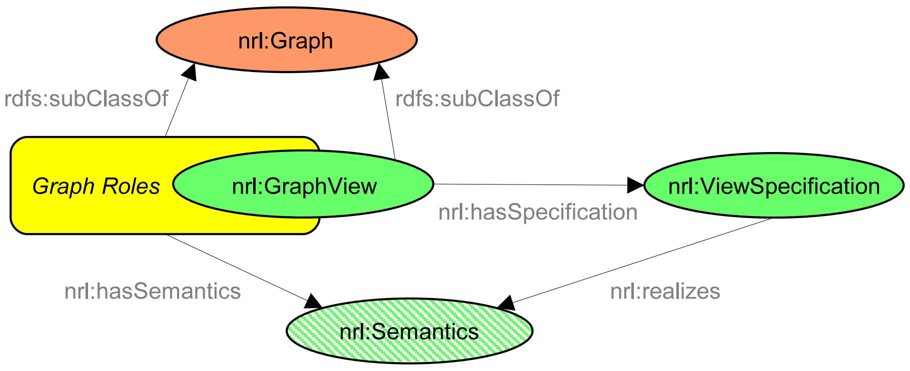


Fig. 5. Graph Views in NRL

which the view is being generated is linked by `nrl:viewOn`. The separation between different interpretations of a named graph and the original named graph itself is thus retained.

nrl:ViewSpecification. This class represents a general view specification, which can currently take one of two forms, modeled as the two subclasses `nrl:RuleViewSpecification` and `nrl:ExternalViewSpecification`. As discussed earlier, semantic views realize procedural semantics and are linked to some semantics via `nrl:realizes`. This is however to be differentiated from `nrl:hasSemantics`, which states that a named graph carries (through a role) declarative semantics which is not necessarily (explicitly) realized via a view specification.

nrl:RuleViewSpecification. Views can be specified by referring to a rule language (via `nrl:ruleLanguage`) and a corresponding set of given rules (via `nrl:rule`). These views are realized by executing the rules, generating the required output named graph.

nrl:ExternalViewSpecification. Instances of this class map to the location of (via `nrl:externalRealizer`) an external application, service, or program that is executed to create the view.

4.6 Example: NRL in Use

In this section, we demonstrate the utilization of the various NRL concepts in a more complex scenario: Ella is a biologist and works as a senior researcher at Institute Pasteur in central Paris. She would like to compile an online knowledge base describing animal species for her students to access. She knows that a rather generic ontology describing the animal species domain, O_1, is already available (which, technically speaking, means it exists as a named graph). Someone else had also supplied data consisting of a vast amount of instances for the animals ontology as a named graph with the role of instance base, I_1. However this combined data does not provide extensive coverage of the animal kingdom as required by Ella. Therefore Ella hires a SW knowledge engineer to model another ontology that defines further species not captured in O_1, and this is stored as another named graph, O_2. Since Ella requires concepts from both ontologies, the engineer merges O_1 and O_2 in the required conceptualization by creating a named graph O as an ontology and defining it as supergraph of O_1 and O_2. Furthermore, a number of real instances of the new animal species defined in O_2 is compiled in an instance base, I_2.

Ella now requires to use all the acquired and generated data to power a useful service for the students to use. Schematic data from the graph O, and the instances from I_1 and I_2 are all imported to a new graph, KB, acting as a knowledge base. Ella would like the students to be able to query the knowledge base with questions like 'Are flatworms Deuterostomes or Platyzoa?'. Although by traversing the animals hierarchy it is clear that they are Platyzoa, the statement is not innately part of the graph KB. This can be discovered by realizing the semantics of `rdfs:subClassOf` as defined in the RDFS semantics. However KB might be required as is, with no assumed semantics, for other purposes. Directly enriching KB with entailment triples permanently would make this impossible.

Therefore the knowledge engineer creates a view over KB for Ella, consisting of the required extended graph, without modifying the original KB in any way. This is done by defining a view specification that computes the procedural semantics for KB. The specification uses a rule language of choice that provides a number of rules, one of which computes the transitive closure of `rdfs:subClassOf` for a set of RDF triples. Executing that rule over the triples in KB results in the semantic view $V_1(KB)$, which consists of the RDF triples in KB plus the generated entailment triples. The separation between the underlying model and the model with the required semantics is thus retained and through simple queries over $V_1(KB)$, students can instantly get answers to their questions.

Ella later on decides to provide another service for younger students by using 'Graph Taxonomy Extractor', a graph visualization API that generates an interactive graph depicting the animal hierarchy within $V_1(KB)$. However this graph contains other information in addition to that required (*e. g.*, properties attributed to classes). Of course, Ella does not want to discard all this useful information from $V_1(KB)$ permanently just to generate the visualization. The knowledge engineer is aware of a *Semantic Web* application that does exactly what Ella requires. The application acts as an external view specification and generates a view, consisting of only triples defining the class hierarchy, over an input named graph. The view generated by this application, $V_2(V_1(KB))$, is fed to the API to effectively generate the interactive graph for the students to explore.

It is worth to note that all seven named graphs on which this last view is generated upon are still intact and have not been affected by any of the operations along the way. If the knowledge engineer requires to apply some different semantics over KB, it may still be done since generating $V_1(KB)$ did not have an impact on KB. However, the content of KB needs to be validated, or generated, each time it is used since one of its subgraphs (O_1, O_2, I_1 and I_2) can change. Although from a practical point of view this might sound laborious, from a conceptual point of view it solves problems regarding data consistency and avoids other problems like working with outdated data that can't be updated because links to underlying models have been lost.

Fig. 6 presents the "dataflow" in our example scenario, demonstrating how the theoretical basis of NRL can be applied in practice to effectively model data for use in different scenarios in a clear and consistent way.

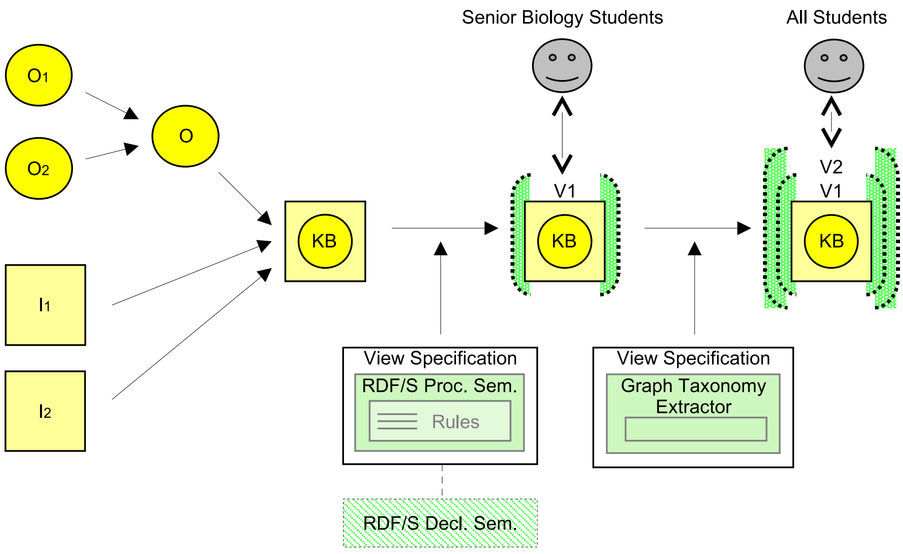


Fig. 6. NRL Dataflow Diagram

We now model the dataflow in Fig. 6 in TriG syntax.[22] TriG is a straightforward extension of Turtle.[23] Turtle itself is an extension of N-Triples[24] which carefully takes the most useful and appropriate things added from Notation3[25] while keeping it in the RDF model. TriG is a plain text format created for serializing NGs and RDF Datasets. Fig. 7 demonstrates how one can make use of the named graph paradigm and the syntax for named graphs:

- `[1]` namespace declarations
- `[2-5]` ontology graphs (`ex:o1` and `ex:o2` are defined and then imported into `ex:o`)
- `[6-8]` instance/knowledge base definitions
- `[9]` contents of ontology `ex:o2`, defining extended animal domain

[22] `http://sites.wiwiss.fu-berlin.de/suhl/bizer/TriG/`

[23] `http://www.dajobe.org/2004/01/turtle/`

[24] `http://www.w3.org/TR/rdf-testcases/#ntriples`

[25] `http://www.w3.org/DesignIssues/Notation3`

```
[1] @prefix nrl: <http://semanticdesktop.org/ontology/nrl-yyyymmdd#> .
    @prefix ex: <http://www.example.org/vocabulary#> .
[2] ex:o2 rdf:type nrl:Ontology .
[3] <http://www.domain.com/o1.rdfs> rdf:type nrl:Ontology ,
     nrl:DocumentGraph .
[4] ex:o1 rdf:type nrl:Ontology ;
         nrl:equivalentGraph <http://www.domain.com/o1.rdfs> .
[5] ex:o rdf:type nrl:Ontology ;
         nrl:imports ex:o1, ex:o2 .
[6] ex:i2 rdf:type nrl:InstanceBase ;
         nrl:hasOntology ex:o2 .
[7] http://www.anotherdomain.com/i1.rdf> rdf:type nrl:InstanceBase,
                                                  nrl:DocumentGraph .
[8] ex:kb rdf:type nrl:KnowledgeBase ;
      nrl:imports ex:o, ex:i2, <http://www.anotherdomain.com/i1.rdf> .
[9] ex:o2 {
      ex:Animal rdf:type rdfs:Class .
             ## further Animal Ontology definitions here ## }
[10]ex:i2 {
     ex:CandyCaneWorm rdf:type ex:Flatworm ;
             ## further Animal Instance definitions here ## }
[11] ex:v1kb rdf:type nrl:KnowledgeBase, nrl:GraphView ;
            nrl:viewOn ex:kb ; nrl:superGraphOf ex:kb ;
            nrl:hasSpecification ex:rvs .
[12] ex:rvs rdf:type nrl:RuleViewSpecification ;
            nrl:realizes ex:RDFSSemantics ; nrl:ruleLanguage "SPARQL" ;
            nrl:rule "CONSTRUCT {?s rdfs:subClassOf ?v} WHERE ..." ;
            nrl:rule "CONSTRUCT {?s rdf:type ?v} WHERE ..." .
[13] ex:RDFSSemantics rdf:type nrl:Semantics ; rdfs:label "RDFS" ;
            nrl:semanticsDefinedBy "http://www.w3.org/TR/rdf-mt/" .
[14] ex:v2v1kb rdf:type nrl:GraphView, nrl:KnowledgeBase ;
            nrl:viewOn ex:v1kb ; nrl:hasSpecification ex:evs .
[15] ex:evs rdf:type nrl:ExternalViewSpecification ;
            nrl:externalRealizer "GraphTaxonomyExtractor" .
```

Fig. 7. NRL Example—TriG Serialization

`[10]` contents of instance base `ex:i2`, defining instances of animals in (`ex:o2`

`[11-13]` `ex:v1kb` is defined as a view on `ex:kb` via the view specification `ex:rvs`; furthermore, `ex:v1kb` is a super graph of `ex:kb` as it realizes the RDFS semantics and thus contains the original graph plus the inferred triples; the view specification is realized (as an example) with some SPARQL-inspired `CONSTRUCT` queries (for this to work, a real rule language is required)

`[14-15]` similar to `[11-13]`, but here we define `ex:v2v1kb` with the help of an external tool, the "GraphTaxonomyExtractor"

5 NEPOMUK Ontologies

Being the representational language, NRL (see Sect. 4) serves as the language required to define the vocabulary with which other lower-level Nepomuk ontologies are represented. All these ontologies can be understood as being instances of NRL, which conceptually is to be found at the representational layer of the Nepomuk ontologies. As such we differentiated between three main ontology layers, as depicted in the Ontologies Pyramid Fig. 8, in order of decreasing generality, abstraction and stability:

1. Representational Layer
2. Upper Level Layer
3. Lower Level Ontologies

Other examples of vocabularies/schemas in the uppermost layer are RDF/S and OWL. Whereas the representational ontologies include abstract high-level classes and properties, constraints, *etc.*; upper-level ontologies provide a framework by which disparate systems may utilize a common knowledge base and from which more domain-specific ontologies may be derived. They are high-level, domain-independent ontologies, characterized by their representation of common sense concepts, *i. e.*, those that are basic for human understanding of the world. Concepts expressed in upper-level ontologies are intended to be basic and universal to ensure generality and expressivity for a wide area of domains. In turn, lower level ontologies (which are further layered into group and personal ontologies) are domain-specific and provide more concrete representations of abstract concepts found in the upper ontologies.

After discussing NRL in detail, we now provide an overview of the engineered ontologies in the upper-level ontology layer. Lower-level ontologies have not been designed by the Nepomuk ontologies task force, but by groups or individuals developing domain-specific applications for the *Social Semantic Desktop*. NRL, together with the upper-level ontologies discussed hereunder, form a central pillar in the *Social Semantic Desktop* system, as they are used to model the environment and domain of the applications. All the described ontologies have been published[26] and although a few of them are open for future adjustments, they are considered to be stable. The modularization of these ontologies in itself was a challenge, especially given the layered approach described earlier. Even though they are all upper-level ontologies, some of them require concepts and/or relationships from others, and therefore dependency relationships exist also within the same level. The design of the upper-level ontologies sought to:

– Represent common desktop entities (objects, people, *etc.*)
– Represent trivial relationships between these entities, as perceived by the desktop user
– Represent a user's mental model, consisting of entities and their relationships on their desktop

[26] http://www.semanticdesktop.org/ontologies/

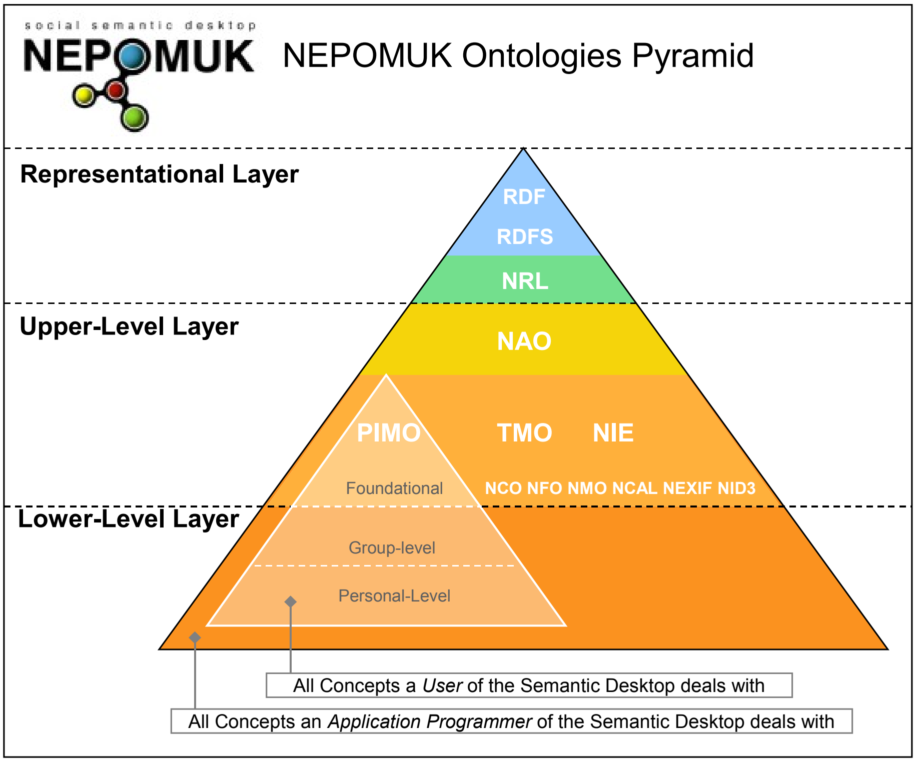

social semantic desktop
NEPOMUK
NEPOMUK Ontologies Pyramid
Representational Layer
Upper-Level Layer
Lower-Level Layer
RDF
RDFS
NRL
NAO
PIMO
TMO
NIE
Foundational
NCO NFO NMO NCAL NEXIF NID3
Group-level
Personal-Level
All Concepts a User of the Semantic Desktop deals with
All Concepts an Application Programmer of the Semantic Desktop deals with

Fig. 8. The Layered Nepomuk Ontologies Pyramid

Whereas the representation of high-level concepts like ‘user’, ‘contact’, ‘desktop’, ‘file’ was fairly straightforward, we also needed to leverage existing information sources in order to make them accessible to semantic applications on the *Social Semantic Desktop*. This information is contained within various structures maintained by the operating system and a multitude of existing ‘legacy’ applications. These structures include specific kinds of entities like messages, documents, pictures, calendar entries and contacts in address books. van Elst coined the term ‘native structures’ to describe them and ‘native resources’ for the pieces of information they contain [25].

A multitude of relationships exist between entities on the desktop. Although in the user’s mind the majority of these relationships is considered trivial (files belonging to the same folder, objects related to the same topic, file copies, professional and social contacts), they remain implicit and undefined. Exposing these relationships to semantic applications is the key to making the desktop truly semantic. The majority of the most basic relationships can be mined through the user’s actions—organizing files in folders, rating desktop objects, saving files from a contact into specific folders, *etc.*Applications on the desktop will also provide means for enabling the user to define less trivial relationships (*e. g.*, through semantic annotation, tagging). During the design of the upper-level ontologies; all kinds of relationships—from the most trivial to ones that are more specific, were taken into account. The user's mental models of how information is stored and organized on their desktop are based on these entities and their relationships—their desktop, files on their desktop, folder structures, contacts who share data, *etc.*The design of a conceptual representation of personal information models was based on the way the users are used to express their knowledge, such that the concepts and relationships in the ontologies reflect the world as seen by the desktop users. This eases the process of knowledge acquisition from the user's desktop structures and activities to their machine-processable representation.

After providing our motivation, in the remaining subsections we will provide an overview of each of the ontologies in the upper-level ontology pyramid layer. The uppermost ontology in the upper-level layer (Fig. 8) is the Nepomuk Annotation Ontology (NAO) [29] which defines trivial relationships between desktop entities as conceived by the user. The Personal Information Model Ontology (PIMO) [26] can be used to express personal information models of individuals, whereas the Task Management Ontology (TMO) [6] is used to describe personal tasks of individuals. The Nepomuk Information Element set of ontologies (NIE) [18] defines common information elements that are to be found on the desktop, together with a number of more specific ontologies whose aim is to represent legacy data in its various forms.

5.1 Nepomuk Annotation Ontology (NAO)

The meaning of the term annotation is highly contextual. Depending on the context, anything can be considered as annotation within a data set (or a named graph). On the SSD, the average user is frequently seen creating representations of objects on their desktop, while the more experienced user is also frequently creating representations of concepts and their relationships. Within this context, we consider annotation to be anything that goes further than creating resources and defining their elementary relationships. A user can create an instance of a 'Person', and provide values for all the elementary properties that an instance of 'Person' can have. The user can then go one step ahead and annotate the resources with more information, of a textual (*e. g.*, custom human-readable descriptions) or non-textual (*e. g.*, links to related resources) nature. In a typical scenario there may be a number of domain-centric properties for the classes 'Person' (*e. g.*, name, address, knows *etc.*) and 'Document' (*e. g.*, author, title, *etc.*). Via vocabulary in the annotation ontology the user can provide personalized, user-friendly labels and descriptions for a resource, as well as additional information like tags and ratings. Generic relationships exist between resources across multiple domains, and making these relationships explicit would be of great benefit for the user. For example, a user may want to state that a 'Document' is about some instance of 'Person'. However this shallow kind of relationship exists between other concepts in other domains. Vocabulary that is able to express these generic relationships are provided by the annotation ontology.

Although this information is optional and does not reflect the elementary nature of a ‘Document’, it contributes to improved data unification and information retrieval via user search.

Graph Metadata is a particular form of annotation, where instead of annotating general resources, one annotates instances of named graphs, *e. g.*, to define the type of graph role. The major difference is that while generic annotation can be stored within any graph the user is working with (*e. g.*, the graph where the annotated resource is defined), metadata about a graph should always be stored outside that graph, in a separate special named graph that is aptly represented in NRL by the nrl:GraphMetadata role. Detailed specifications for this ontology are available in full online [29].

5.2 Nepomuk Information Element (NIE)

The abbreviation NIE may refer to the NIE Ontology Framework as a whole or to the NIE Core Ontology. The motivation for NIE lies in representing the multitude of applications and data formats. Previous semantic desktop projects (*e. g.*, Haystack Haystack [20] or Gnowsis [23]) had to develop their solutions. Some attempts at standardization have been made (*e. g.*, Adobe XMP[27], Freedesktop.org XESAM[28]) but a definite standard had not emerged before NIE’s conception. Apart from large metadata description frameworks there exists a considerable number of smaller single-purpose ontologies aimed at specific types of resources (*e. g.*, ICAL[29] or VCARD[30]). A broad array of utilities has been developed for extracting RDF metadata from desktop sources[31].

Various problems have been identified with the pre-existing vocabularies. They are expressed in many languages and the level of detail often leaves much to be desired. The NIE Framework is an attempt to build upon that experience, to provide unified vocabulary to describe typical native resources that may be of interest to the user. These resources are intended to serve as raw data for other semantic applications. They can be browsed, searched, annotated and linked with each other. Data represented in NIE has three roles. First, NIE data is intended to be generated by an extraction process. Second, RDF-based systems can create NIE structures natively, without building on existing applications. Third, data expressed in NIE can be imported back to native applications. Thus, the resulting ontologies serve as a mediator between semantic and native applications. The full specifications for this ontology can be accessed online [18].

5.3 Personal Information Model Ontology (PIMO)

The scope of PIMO is to model data that is within the attention of the user and needed for knowledge work or private use. The focus is on data that is accessed

[27] http://www.adobe.com/products/xmp/

[28] http://xesam.org/main/XesamAbout

[29] http://www.ietf.org/rfc/rfc2445.txt

[30] http://www.ietf.org/rfc/rfc2426.txt

[31] http://simile.mit.edu/wiki/RDFizers

through a *Semantic Desktop* or other personalized *Semantic Web* applications. We call this the Personal Knowledge Workspace [15] or Personal Space of Information [16], embracing all data needed by an individual to perform knowledge work. Today, such data is typically stored in files, in Personal Information Management or in groupware systems. A user has to cope with different formats of data, such as text documents, contact information, e-mails, appointments, task lists, project plans, or an Enterprise Resource Planning system. Existing information that is already stored in information systems is in the scope of PIMO, but abstract concepts can also be represented, if needed. PIMO is based on the idea that users have a mental model to categorize their environment. Each concept in the environment of the user is represented as a Thing in the model, and mapped to documents and other entities that mention the concept. Things can be described via their relations to other Things or by literal RDF properties.

In PIMO, Things are connected to their equivalent resources using directed relations. The design rationale was to keep the PIMO ontology itself, as well as the data needed to create a PIMO for a user as minimal as possible. Inside a user's PIMO, duplication is avoided. PIMO builds on NRL, NIE and NAO. By addressing all key issues: precise representation, easy adoption, easy to understand by users, extensibility, interoperability, reuse of existing ontologies and data integration; PIMO provides a framework for creating personal information management applications and ontologies. The detailed specifications for this ontology are available online [26].

5.4 Task Model Ontology (TMO)

The TMO is a conceptual representation of tasks for use in personal task management applications for the knowledge worker (KWer). It represents an agreed, domain-specific information model for tasks and covers personal task management use cases. As a domain model the TMO models the tasks a KWer deals with in the context of the KWer's other personal information. It thereby represents an activity-centric view on the KWer's personal information, as it models underlying tasks as well the relations to other personal information that is relevant to that task. The KWer regards all personal information as a single body of information [21], a personal information cloud including tasks. Information-wise, the use cases focus on an individual KWer's personal tasks and further related personal information. The full specifications for this ontology are available online [6] and they presents the state-of-the art in task models and the semi-formal description of the ontology with links to the supported use cases.

6 Summary and Outlook

The *Social Semantic Desktop* as presented in this material provides a universal platform for:

- Personal Information Management
- Distributed Information Management
- Social Expansion and Community Creation

In order to operate, the *Social Semantic Desktop* requires metadata, which can be extracted by:

- i) Lifting of existing structured data onto RDF
- ii) Usage of Human Language Technology (HLT) to capture knowledge from text and transform that into RDF
- ii) Manual creation of metadata by linking, annotation or tagging

In order to soften the border between the *Semantic Web* and the *Social Semantic Desktop*, we have applied *Semantic Web* knowledge representation for the desktop. Aligning knowledge representation on a *Social Semantic Desktop* with the general *Semantic Web* approaches (RDF, RDFS, OWL, *etc.*) promises a comprehensive use of data and schemas and an active, personalized access point to the *Semantic Web* [24]. In such a scenario, ontologies play an important role, from very general ontologies stating which entities can be modeled on a *Semantic Desktop* (*e. g.*, people, documents, *etc.*) to rather personal vocabulary structuring information items. One of the most important design decisions is the question of the *representational ontology*, constraining the general expressivity of such a system. In this paper, we concentrated on those parts of the NEPOMUK Representational Language (NRL) which are rooted in the requirements which arose by the *distributed knowledge representation and heterogeneity aspects* of the *Semantic Desktop* scenario, and which we think cannot satisfactorily be dealt with by the current state of the art. In a nutshell, the basic arguments and design principles of NRL are as follows:

- Due to the heterogeneity of the data-creating and data-consuming entities in the social semantic desktop scenario, a single interpretation schema cannot be assumed. Therefore, NRL aims at a *strict separation between data (sets of triples, graphs) and their interpretation/semantics.*
- Imposing specific semantics to a graph is realized by generating *views* on that graph. Such a generation is directed by an (executable) *view specification* which may realize a declarative semantics (*e. g.*, the RDF/S or OWL semantics specified in a standardization document).
- Graph views cannot only be used for semantic interpretations of graphs, but also for application-driven tailoring of a graph.[32]
- Handling of multiple graphs (with different provenance, ownership, level of trust, *etc.*) is essential. *Named graphs* are the basic means targeting this problem.
- Graphs can play different roles in different contexts. While for one application a graph may be an ontology, another one may see it as plain data. These *roles* can explicitly be specified.

While originally designed as a NEPOMUK internal standard for the *Social Semantic Desktop*, we believe that the arguments also hold for the general *Semantic Web*. This is especially true when we review the current trends which increasingly

[32] This corresponds to a database-like view concept.

show a shift from the view of "the Semantic Web as one big, global knowledge base" to "a Web of (machine and human) actors" with local perspectives and social needs like trust, ownership, *etc.*

Within NEPOMUK, we have developed the approach technically, by complementing the NRL standard with tools that facilitate its use by the application programmer, as well as conceptually, by the development and integration of a number of accompanying ontology standards[33]; *e. g.*, the annotation vocabulary referenced earlier, an information element ontology, and an upper-ontology for *Personal Information Models*.

Acknowledgements. This work was supported by the European Union IST fund (Grant FP6-027705, Project NEPOMUK); by the German Federal Ministry of Education, Science, Research and Technology (bmb+f), (Grant 01 IW F01, Project Mymory: Situated Documents in Personal Information Spaces) and by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2). The authors would especially like to thank all contributors to NEPOMUK's ontology taskforce.

References

1. Austin, J.L.: How to do things with words. Harvard U.P., Cambridge (1962)
2. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinnes, D., Patel-Schneider, P., Stein, L.: OWL web ontology language reference (2004)
3. Beckett, D.: RDF/XML syntax specification (revised). W3C recommendation, W3C (February 2004), http://www.w3.org/TR/rdf-syntax-grammar/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 89 (May 2001)
5. Brickley, D., Guha, R.: RDF vocabulary description language 1.0: RDF Schema. Technical report, W3C (February 2004), http://www.w3.org/TR/rdf-schema/
6. Brunzel, M., Grebner, O.: Nepomuk task model ontology specification. Technical report, NEPOMUK Consortium (2008)
7. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 613–622. ACM Press, New York (2005)
8. Cheyer, A., Park, J., Giuli, R.: Iris: Integrate. relate. infer. share. In: Decker, S., Park, J., Quan, D., Sauermann, L. (eds.) Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland, November 6, vol. 175 (2005)
9. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A framework and graphical development environment for robust NLP tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (2002)
10. Cunningham, H., Maynard, D., Tablan, V.: JAPE: a Java Annotation Patterns Engine (2nd edn.). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield (November 2000)

[33] http://www.semanticdesktop.org/ontologies/

11. Davis, B., Handschuh, S., Cunningham, H., Tablan, V.: Further Use of Controlled Natural Language for Semantic Annotation. In: Proceedings of the 1st Semantic Authoring and Annotation Workshop (SAAW 2006) at ISWC 2006, Athens, Georgia, USA (2006)
12. Davis, B., Iqbal, A., Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Handschuh, S.: RoundTrip Ontology Authoring. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 50–65. Springer, Heidelberg (2008)
13. Decker, S., Frank, M.: The social semantic desktop. In: Proc. of the WWW 2004 Workshop Application Design, Development and Implementation Issues in the Semantic Web (2004)
14. Hayes, P.: RDF semantics. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/rdf-mt/
15. Holz, H., Maus, H., Bernardi, A., Rostanin, O.: From lightweight, proactive information delivery to business process-oriented knowledge management. Journal of Universal Knowledge Management 0(2), 101–127 (2005)
16. Jones, W.P., Teevan, J.: Personal Information Management. University of Washington Press (October 2007)
17. Manola, F., Miller, E.: RDF primer. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/rdf-primer/
18. Mylka, A., Sauermann, L., Sintek, M., van Elst, L.: Nepomuk information element framework specification. Technical report, NEPOMUK Consortium (2007)
19. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C working draft, W3C (2005), http://www.w3.org/TR/rdf-sparql-query/
20. Quan, D., Huynh, D., Karger, D.R.: Haystack: A platform for authoring end user semantic web applications. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 738–753. Springer, Heidelberg (2003)
21. Ravasio, P., Tscherter, V.: Users' theories of the desktop metaphor, or why we should seek metaphor-free interfaces. In: Kaptelinin, V., Czerwinski, M. (eds.) Beyond the desktop metaphor: designing integrated digital work environments, pp. 265–294. MIT Press, Cambridge (2007)
22. Richter, J., Völkel, M., Haller, H.: DeepaMehta – A Semantic Desktop. In: Decker, S., Park, J., Quan, D., Sauermann, L. (eds.) Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland, November 6, vol. 175 (2005)
23. Sauermann, L.: The gnowsis—using semantic web technologies to build a semantic desktop. Diploma thesis, Technical University of Vienna (2003)
24. Sauermann, L., Dengel, A., Elst, L., Lauer, A., Maus, H., Schwarz, S.: Personalization in the EPOS project. In: Bouzid, M., Henze, N. (eds.) Proceedings of the International Workshop on Semantic Web Personalization, Budva, Montenegro, June 12, pp. 42–52 (2006)
25. Sauermann, L., van Elst, L., Dengel, A.: PIMO—a framework for representing personal information models. In: Tochtermann, K., Haas, W., Kappe, F., Scharl, A., Pellegrini, T., Schaffert, S. (eds.) Proceedings of I-MEDIA 2007 and I-SEMANTICS 2007 (2007)
26. Sauermann, L., van Elst, L., Moeller, K.: Nepomuk personal information model ontology specification. Technical report, NEPOMUK Consortium (2007)
27. Scerri, S., Handschuh, S., Decker, S.: Semantic Email as a Communication Medium for the Social Semantic Desktop. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 124–138. Springer, Heidelberg (2008)

28. Scerri, S., Mencke, M., Davis, B., Handschuh, S.: Evaluating the Ontology underlying sMail - the Conceptual Framework for Semantic Email Communication. In: Proceedings of the 6th International conference of Language Resources and Evaluation (LREC), Marrakech, Morocco (2008)
29. Scerri, S., Sintek, M., van Elst, L., Handschuh, S.: Nepomuk annotation ontology specification. Technical report, NEPOMUK Consortium (2007)
30. Schenk, S., Staab, S.: Networked graphs: A declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In: Proceedings of the 17th International World Wide Web Conference, Bejing, China (2008)
31. Searle, J.R.: Speech Acts. Cambridge University Press, Cambridge (1969)
32. Sintek, M., Decker, S.: *TRIPLE*–A query, inference, and transformation language for the semantic web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, p. 364. Springer, Heidelberg (2002)
33. Sintek, M., van Elst, L., Scerri, S., Handschuh, S.: Distributed knowledge representation on the social semantic desktop: Named graphs, views and roles in NRL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 594–608. Springer, Heidelberg (2007)
34. Sintek, M., van Elst, L., Grimnes, G., Scerri, S., Handschuh, S.: Knowledge representation for the distributed, social semantic web: Named graphs, graph roles and views in nrl. In: Cuenca-Grau, B., Honavar, V., Schlicht, A., Wolter, F. (eds.) Second International Workshop on Modular Ontologies, WoMO 2007 (2007)
35. Thai, V., Handschuh, S., Decker, S.: IVEA: An information visualization tool for personalized exploratory document collection analysis. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 139–153. Springer, Heidelberg (2008)
36. Thai, V., Handschuh, S., Decker, S.: Tight coupling of personal interests with multi-dimensional visualization for exploration and analysis of text collections. In: IV 2008: Proceedings of the 12th International Conference on Information Visualisation, pp. 221–226. IEEE Computer Society, Los Alamitos (2008)
37. van Elst, L., Dignum, V., Abecker, A.: Towards agent-mediated knowledge management. In: van Elst, L., Dignum, V., Abecker, A. (eds.) AMKM 2003. LNCS (LNAI), vol. 2926, pp. 1–31. Springer, Heidelberg (2004)

Ontologies and Databases: The *DL-Lite* Approach

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Maurizio Lenzerini[2], Antonella Poggi[2], Mariano Rodriguez-Muro[1], and Riccardo Rosati[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano, Italy
{calvanese,rodriguez}@inf.unibz.it
[2] Dipartimento di Informatica e Sistemistica
SAPIENZA Università di Roma, Italy
lastname@dis.uniroma1.it

Abstract. Ontologies provide a conceptualization of a domain of interest. Nowadays, they are typically represented in terms of Description Logics (DLs), and are seen as the key technology used to describe the semantics of information at various sites. The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular, but for it to become widespread in standard applications, it is fundamental that the conceptual layer through which the underlying data layer is accessed does not introduce a significant overhead in dealing with the data. Based on these observations, in recent years a family of DLs, called *DL-Lite*, has been proposed, which is specifically tailored to capture basic ontology and conceptual data modeling languages, while keeping low complexity of reasoning and of answering complex queries, in particular when the complexity is measured w.r.t. the size of the data. In this article, we present a detailed account of the major results that have been achieved for the *DL-Lite* family. Specifically, we concentrate on $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$, an expressive member of this family, present algorithms for reasoning and query answering over $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$ ontologies, and analyze their computational complexity. Such algorithms exploit the distinguishing feature of the logics in the *DL-Lite* family, namely that ontology reasoning and answering unions of conjunctive queries is first-order rewritable, i.e., it can be delegated to a relational database management system. We analyze also the effect of extending the logic with typical DL constructs, and show that for most such extensions, the nice computational properties of the *DL-Lite* family are lost. We address then the problem of accessing relational data sources through an ontology, and present a solution to the notorious impedance mismatch between the abstract objects in the ontology and the values appearing in data sources. The solution exploits suitable mappings that create the objects in the ontology from the appropriate values extracted from the data sources. Finally, we discuss the QUONTO system that implements all the above mentioned solutions and is wrapped by the DIG-QUONTO server, thus providing a standard DL reasoner for $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$ with extended functionality to access external data sources.

1 Introduction

An *ontology* is a formalism whose purpose is to support humans or machines to share some common knowledge in a structured way. Guarino [44] distinguishes *Ontology*, the discipline that studies the nature of being, from *ontologies* (written with lower-case initial) that are systems of categories that account for a certain view or aspect of the world. Such ontologies act as standardized reference models to support knowledge sharing and integration, and with respect to this their role is twofold: (*i*) they support human understanding and communication, and (*ii*) they facilitate content-based access, communication, and integration across different information systems; to this aim, it is important that the language used to express ontologies is formal and machine-processable. To accomplish such tasks, an ontology must focus on the explication and formalization of the *semantics* of enterprise application information resources and of the relationships among them. According to Gruber [43,42], an ontology is a formal, explicit specification of a shared conceptualization. A *conceptualization* is an abstract representation of some aspect of the world (or of a fictitious environment) which is of interest to the users of the ontology. The term *explicit* in the definition refers to the fact that constructs used in the specification must be explicitly defined and the users of the ontology, who share the information of interest and the ontology itself, must agree on them. *Formal* means that the specification is encoded in a precisely defined language whose properties are well known and understood; usally this means that the languages used for the specification of an ontology is logic-based, such as the languages used in the Knowledge Representation and Artificial Intelligence communities. *Shared* means that the ontology is meant to be shared across several people, applications, communities, and organizations. According to the W3C Ontology Working Group[1], an ontology defines a set of representational *terms* used to describe and represent an area of knowledge. The ontology can be described by giving the semantics to such terms [43]. More specifically, such terms, also called *lexical references*, are associated with (i.e., mapped to) entities in the domain of interest; formal axioms are introduced to precisely state such mappings, which are in fact the statements of a logical theory. In other words, an ontology is an explicit representation of the semantics of the domain data [65]. To sum up, though there is no precise common agreement on what an ontology is, there is a common core that underlies nearly all approaches [88]:

– a vocabulary of terms that refer to the things in the domain of interest;
– a specification of the meaning (semantics) of the terms, given (ideally) in some sort of formal logics.

Some simple ontologies consist only of a mere *taxonomy* of terms; however, usually ontologies are based on rigorous logical theories, equipped with reasoning algorithms and services. According to Gruber [43,42], knowledge in ontologies is mainly formalized using five kinds of components:

1. *concepts (or classes)*, which represent sets of objects with common properties within the domain of interest;
2. *relations*, which represent relationships among concepts by means of the notion of mathematical relation;
3. *functions*, which are functional relations;

[1] `http://www.w3c.org/2001/sw/WebOnt/`

4. *axioms (or assertions)*, which are sentences that are always true and are used in general to enforce suitable properties of classes, relations, and individuals;
5. *individuals (or instances)*, which are individual objects in the domain of interest.

Ontologies allow the key concepts and terms relevant to a given domain to be identified and defined in an unambiguous way. Moreover, ontologies facilitate the integration of different perspectives, while capturing key distinctions in a given perspective; this improves the cooperations of people or services both within a single organization and across several organizations.

1.1 Ontologies vs. Description Logics

An ontology, as a conceptualization of a domain of interest, provides the mechanisms for modeling the domain and reasoning upon it, and has to be represented in terms of a well-defined language. Description Logics (DLs) [7] are logics specifically designed to represent structured knowledge and to reason upon it, and as such are perfectly suited as languages for representing ontologies. Given a representation of the domain of interest, an ontology-based system should provide well-founded methods for reasoning upon it, i.e., for analyzing the representation, and drawing interesting conclusions about it. DLs, being logics, are equipped with reasoning methods, and DL-based systems provide reasoning algorithms and working implementations for them. This explains why variants of DLs are providing now the underpinning for the ontology languages promoted by the W3C, namely the standard Web Ontology Language OWL[2] and its variants (called *profiles*), which are now in the process of being standardized by the W3C in their second edition, OWL 2.

DLs stem from the effort started in the mid 80s to provide a formal basis, grounded in logic, to formalisms for the structured representation of knowledge that were popular at that time, notably Semantic Networks and Frames [67,14], that typically relied on graphical or network-like representation mechanisms. The fundamental work by Brachman and Levesque [12], initiated this effort, by showing on the one hand that the full power of First-Order Logic is not required to capture the most common representation elements, and on the other hand that the computational complexity of inference is highly sensitive to the expressive power of the KR language. Research in DLs up to our days can be seen as the systematic and exhaustive exploration of the corresponding tradeoff between expressiveness and efficiency of the various inference tasks associated to KR.

DLs are based on the idea that the knowledge in the domain to represent should be structured by grouping into classes objects of interest that have properties in common, and explicitly representing those properties through the relevant relationships holding among such classes. *Concepts* denote classes of objects, and *roles* denote (typically binary) relations between objects. Both are constructed, starting from atomic concepts and roles, by making use of various constructs, and it is precisely the set of allowed constructs that characterizes the (concept) language underlying a DL.

The domain of interest is then represented by means of a DL *knowledge base* (KB), where a separation is made between general intensional knowledge and specific knowledge about individual objects in the modeled domain. The first kind of knowledge is

[2] `http://www.w3.org/2007/OWL/`

maintained in what has been traditionally called a *TBox* (for "Terminological Box"), storing a set of universally quantified assertions that state general properties of concepts and roles. The latter kind of knowledge is represented in an *ABox* (for "Assertional Box"), constituted by assertions on individual objects, e.g., the one stating that an individual is an instance of a certain concept.

Several reasoning tasks can be carried out on a DL KB, where the basic form of reasoning involves computing the subsumption relation between two concept expressions, i.e., verifying whether one expression always denotes a subset of the objects denoted by another expression. More in general, one is interested in understanding how the various elements of a KB interact with each other in an often complex way, possibly leading to inconsistencies that need to be detected, or implying new knowledge that should be made explicit.

The above observations emphasize that a DL system is characterized by three aspects: (*i*) the set of constructs constituting the language for building the concepts and the roles used in a KB; (*ii*) the kind of assertions that may appear in the KB; (*iii*) the inference mechanisms provided for reasoning on the knowledge bases expressible in the system. The expressive power and the deductive capabilities of a DL system depend on the various choices and assumptions that the system adopts with regard to the above aspects. In the following, we present .

1.2 Expressive Power vs. Efficiency of Reasoning in Description Logics

The first aspect above, i.e., the language for concepts and roles, has been the subject of an intensive research work started in the late 80s. Indeed, the initial results on the computational properties of DLs have been devised in a simplified setting where both the TBox and the ABox are empty [69,84,38]. The aim was to gain a clear understanding of the properties of the language constructs and their interaction, with the goal of singling out their impact on the complexity of reasoning. Gaining this insight by understanding the combinations of language constructs that are *difficult* to deal with, and devising general methods to cope with them, is essential for the design of inference procedures. It is important to understand that in this context, the notion of "difficult" has to be understood in a precise technical sense, and the declared aim of research in this area has been to study and understand the frontier between tractability (i.e., solvable by a polynomial time algorithm) and intractability of reasoning over concept expressions. The maximal combinations of constructs (among those most commonly used) that still allowed for polynomial time inference procedures were identified, which allowed to exactly characterize the tractability frontier [38]. It should be noted that the techniques and technical tools that were used to prove such results, namely tableaux-based algorithms, are still at the basis of the modern state of the art DL reasoning systems [68], such as Fact [49], Racer [45], and Pellet [86,85].

The research on the tractability frontier for reasoning over concept expressions proved invaluable from a theoretical point of view, to precisely understand the properties and interactions of the various DL constructs, and identify practically meaningful combinations that are computationally tractable. However, from the point of view of knowledge representation, where knowledge about a domain needs to be encoded, maintained, and reasoned upon, the assumption of dealing with concept expressions only, without considering a KB (i.e., a TBox and possibly an ABox) to which the concepts refer, is clearly unrealistic. Early successful DL KR systems, such as Classic [74], relied on a KB, but did not renounce to tractability by imposing syntactic restrictions on the use of concepts in definitions, essentially to ensure acyclicity (i.e., lack of mutual recursion). Under such an assumption, the concept definitions in a KB can be folded away, and hence reasoning over a KB can be reduced to reasoning over concept expressions only.

However, the assumption of acyclicity is strongly limiting the ability to represent real-world knowledge. These limitations became quite clear also in light of the tight connection between DLs and formalisms for the structured representation of information used in other contexts, such as databases and software engineering [31]. In the presence of cyclic KBs, reasoning becomes provably exponential (i.e, ExpTime-complete) already when the concept language contains rather simple constructs. As a consequence of such a result, research in DLs shifted from the exploration of the tractability border to an exploration of the decidability border. The aim has been to investigate how much the expressive power of language and knowledge base constructs could be further increased while maintaining decidability of reasoning, possibly with the same, already rather high, computational complexity of inference. The techniques used to prove decidability and complexity results for expressive variants of DLs range from exploiting the correspondence with modal and dynamic logics [83,30], to automata-based techniques [92,91,26,28,18,8], to tableaux-based techniques [6,15,52,9,53]. It is worth noticing that the latter techniques, though not computationally optimal, are amenable to easier implementations, and are at the basis of the current state-of-the-art reasoners for expressive DLs [68].

1.3 Accessing Data through Ontologies

Current reasoners for expressive DLs perform indeed well in practice, and show that even procedures that are exponential in the size of the KB might be acceptable under suitable conditions. However, such reasoners have not specifically been tailored to deal with large amounts of data (e.g., a large ABox). This is especially critical in those settings where ontologies are used as a high-level, conceptual view over data repositories, allowing users to access data item without the need to know how the data is actually organized and where it is stored. Typical scenarios for this that are becoming more and more popular are those of Information and Data Integration Systems [63,70,29,41], the Semantic Web [47,51], and ontology-based data access [37,75,20,48]. Since the common denominator to all these scenarios, as far as this article is concerned, is the access to data through an ontology, we will refer to them together as *Ontology-Based Data Access* (OBDA).

In OBDA, data are typically very large and dominate the intentional level of the ontologies. Hence, while one could still accept reasoning that is exponential on the intentional part, it is mandatory that reasoning is polynomial (actually less – see later) in the data. If follows that, when measuring the computational complexity of reasoning, the most important parameter is the size of the data, i.e., one is interested in so-called *data complexity* [90]. Traditionally, research carried out in DLs has not paid much attention to the data complexity of reasoning, and only recently efficient management of large amounts of data [50,33] has become a primary concern in ontology reasoning systems, and data-complexity has been studied explicitly [55,22,71,62,3,4]. Unfortunately, research on the trade-off between expressive power and computational complexity of reasoning has shown that many DLs with efficient reasoning algorithms lack the modeling power required for capturing conceptual models and basic ontology languages. On the other hand, whenever the complexity of reasoning is exponential in the size of the instances (as for example for the expressive fragments of OWL and OW2, or in [27]), there is little hope for effective instance management.

A second fundamental requirement in OBDA is the possibility to answer queries over an ontology that are more complex than the simple queries (i.e., concepts and roles) usually considered in DLs research. It turns out, however, that one cannot take the other extreme and adopt as a query language full SQL (corresponding to First-Order Logic queries), since due to the inherent incompleteness introduced by the presence of an ontology, query answering amounts to logical inference, which is undecidable for First-Order Logic. Hence, a good trade-off regarding the query language to use can be found by considering those query languages that have been advocated in databases in those settings where incompleteness of information is present [89], such as data integration [63] and data exchange [57,64]. There, the query language of choice are conjunctive queries, corresponding to the select-project-join fragment of SQL, and unions thereof, which are also the kinds of queries that are best supported by commercial database management systems.

In this paper we advocate that for OBDA, i.e., all for those contexts where ontologies are used to access large amounts of data, a suitable DL should be used, specifically tailored to capture all those constructs that are used typically in conceptual modeling, while keeping query answering efficient. Specifically, efficiency should be achieved by delegating data storage and query answering to a relational data management systems (RDBMS), which is the only technology that is currently available to deal with complex queries over large amounts of data. The chosen DL should include the main modeling features of conceptual models, which are also at the basis of most ontology languages. These features include cyclic assertions, ISA and disjointness of concepts and roles, inverses on roles, role typing, mandatory participation to roles, functional restrictions of roles, and a mechanisms for identifying instances of concepts. Also, the query language should go beyond the expressive capabilities of concept expressions in DLs, and allow for expressing conjunctive queries and unions thereof.

1.4 Preliminaries on Computational Complexity

In the following, we will assume that the reader is familiar with basic notions about computational complexity, as defined in standard textbooks [40,73,61]. In particular, we will refer to the following complexity classes:

$$\mathrm{AC}^0 \subsetneq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{PTime} \subseteq \textsc{NP} \subseteq \textsc{ExpTime}.$$

We have depicted the known relationships between these complexity classes. In particular, it is known that AC^0 is strictly contained in LogSpace, while it is open whether any of the other depicted inclusions is strict. However, it is known that PTime $\subsetneq$ ExpTime.

Also, we will refer to the complexity class coNP, which is the class of problems that are the complement of a problem in NP.

We only comment briefly on the complexity classes AC^0, LOGSPACE, and NLOGSPACE, since readers might be less familiar with them.

A (decision) problem belongs to LOGSPACE if it can be decided by a two-tape (deterministic) Turing machine that receives its input on the read-only input tape and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input. The complexity class NLOGSPACE is defined analogously, except that a non-deterministic Turing machine is used instead of a deterministic one. A typical problem that is in LOGSPACE (but not in AC^0) is undirected graph reachability [78]. A typical problem that is in NLOGSPACE is directed graph reachability.

A LOGSPACE *reduction* is a reduction computable by a three-tape Turing machine that receives its input on the read-only input tape, leaves its output on the write-only output tape, and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input. We observe that most reductions among decision problems presented in the computer science literature, including all reductions that we present here, are actually LOGSPACE reductions.

For the complexity class AC^0, we provide here only the basic intuitions, and refer to [93] for the formal definition, which is based on the circuit model. Intuitively, a problem belongs to AC^0 if it can be decided in constant time using a number of processors that is polynomial in the size of the input. A typical example of a problem that belongs to AC^0 is the evaluation of First-Order Logic (i.e., SQL) queries over relational databases, where only the database is considered to be the input, and the query is considered to be fixed [1]. This fact is of importance in the context of what discussed in this paper, since the low complexity in the size of the data of the query evaluation problem provides an intuitive justification for the ability of relational database engines to deal efficiently with very large amounts of data. Also, whenever a problem is shown to be hard for a complexity class that *strictly* contains AC^0 (such as LOGSPACE and all classes above it), then it cannot be reduced to the evaluation of First-Order Logic queries (cf. Section 2.6).

1.5 Overview of This Article

We start by presenting a family of DLs, called *DL-Lite*, that has been proposed recently [21,22,24] with the aim of addressing the above issues. Specifically, in Section 2, we present *DL-Lite*$_{\mathcal{A},id}$, a significant member of the *DL-Lite* family. One distinguishing feature of *DL-Lite*$_{\mathcal{A},id}$ is that it is tightly related to conceptual modeling formalisms and is actually able to capture their most important features, as illustrated for UML class diagrams in Section 3.

A further distinguishing feature of *DL-Lite*$_{\mathcal{A},id}$ is that query answering over an ontology can be performed as a two step process: in the first step, a query posed over the ontology is reformulated, taking into account the intensional component (the TBox) only, obtaining a union of conjunctive queries; in the second step such a union is directly evaluated over the extensional component of the ontology (the ABox). Under the assumption that the ABox is maintained by an RDBMS in secondary storage, the evaluation can be carried out by an SQL engine, taking advantage of well established query optimization strategies. Since the first step does not depend on the data, and the second step is the evaluation of a relational query over a databases, the whole query answering process is in AC^0 in the size of the data [1], i.e., it has the same complexity as the plain evaluation of a conjunctive query over a relational database. In Section 4, we discuss the traditional DL reasoning services for *DL-Lite*$_{\mathcal{A},id}$ and show that they are polynomial in the size of the TBox, and in AC^0 in the size of the ABox (i.e., the data). Then, in Section 5 we discuss query answering and its complexity.

We show also, in Section 6, that *DL-Lite*$_{\mathcal{A},id}$ is essentially the maximal fragment exhibiting such desirable computational properties, and allowing one to ultimately delegate query answering to a relational engine [22,20,25]. Indeed, even slight extensions of *DL-Lite*$_{\mathcal{A},id}$ make query answering (actually already instance checking, i.e., answering atomic queries) at least NLogSpace-hard in data complexity, ruling out the possibility that query evaluation could be performed by a relational engine.

Finally, we address the issue that the TBox of the ontology provides an abstract view of the intensional level of the application domain, whereas the information about the extensional level (the instances of the ontology) reside in the data sources, which are developed independently of the conceptual layer, and are managed by traditional technologies (e.g., a relational database). Therefore, the problem arises of establishing sound mechanisms for linking existing data to the instances of the concepts and the roles in the ontology. We present, in Section 7, a recently proposed solution for this problem [75], based on a mapping mechanism to link existing data sources to an ontology expressed in *DL-Lite*$_{\mathcal{A},id}$. Such mappings allow one also to bridge the notorious *impedance mismatch problem* between values (data) stored in the sources and abstract objects that are instances of concepts and roles in the ontology [66]. Intuitively, the objects in the ontology are generated by the mappings from the data values retrieved from the data sources, by making use of suitable (designer defined) skolem functions.

All the reasoning and query answering techniques presented in the paper have been implemented in the QuOnto system [2,76], and have been wrapped in the DIG-QuOnto tool to provide a standard interface for DL reasoners according to the DIG protocol. This is discussed in Section 8, together with a Plugin for the ontology editor Protégé that provides functionalities for ontology-based data access. Finally, in Section 9, we conclude the paper.

We point out that most of the work presented in this article has been carried out within the 3-year EU FET STREP project "Thinking ONtologiES" (TONES)[3]. We also remark that large portions of the material in Sections 2, 4, and 5 are inspired or taken from [24,25], of the material in Section 6 from [22,20], and of the material in Section 7 from [75]. However, the notation and formalisms have been unified, and proofs have been revised and extended to take into account the proper features of the formalism considered here, which in part differs from the ones considered in the above mentioned works.

2 The Description Logic *DL-Lite*$_{\mathcal{A},id}$

In this section, we introduce formally syntax and semantics of DLs, and we do so for *DL-Lite*$_{\mathcal{A},id}$, a specific DL of the *DL-Lite* family [24,22], that is also equipped with

[3] `http://www.tonesproject.org/`

identification constraints [24]. We will show in the subsequent chapters that in *DL-Lite*$_{\mathcal{A},id}$ the trade-off between expressive power and computational complexity of reasoning is optimized towards the needs that arise in ontology-based data access. In other words, *DL-Lite*$_{\mathcal{A},id}$ is able to capture the most significant features of popular conceptual modeling formalisms, nevertheless query answering can be managed efficiently by relying on relational database technology.

2.1 *DL-Lite*$_{\mathcal{A},id}$ Expressions

As mentioned, in Description Logics [7] (DLs) the domain of interest is modeled by means of *concepts*, which denote classes of objects, and *roles* (i.e., binary relationships), which denote binary relations between objects. In addition, *DL-Lite*$_{\mathcal{A},id}$ distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. We now define formally syntax and semantics of expressions in our logic.

Like in any other logic, *DL-Lite*$_{\mathcal{A},id}$ expressions are built over an alphabet. In our case, the alphabet comprises symbols for atomic concepts, value-domains, atomic roles, atomic attributes, and constants.

Syntax. The value-domains that we consider in *DL-Lite*$_{\mathcal{A},id}$ are those corresponding to the data types adopted by the Resource Description Framework (RDF)[4], such as `xsd:string`, `xsd:integer`, etc. Intuitively, these types represent sets of values that are pairwise disjoint. In the following, we denote such value-domains by $T_1, \ldots, T_n$. Furthermore, we denote with Γ the alphabet for constants, which we assume partitioned into two sets, namely, Γ_O (the set of constant symbols for *objects*), and Γ_V (the set of constant symbols for *values*). In turn, Γ_V is partitioned into n sets $\Gamma_{V_1}, \ldots, \Gamma_{V_n}$, where each Γ_{V_i} is the set of constants for the values in the value-domain T_i.

In providing the specification of our logic, we use the following notation:

1. A denotes an *atomic concept*, B a *basic concept*, C a *general concept*, and $\top_c$ the *universal concept*. An atomic concept is a concept denoted by a name. Basic and general concepts are concept expressions whose syntax is given at point 1 below.
2. E denotes a *basic value-domain*, i.e., the range of an attribute, F a *value-domain expression*, and $\top_d$ the *universal value-domain*. The syntax of value-domain expressions is given at point 2 below.
3. P denotes an *atomic role*, Q a *basic role*, and R a *general role*. An atomic role is simply a role denoted by a name. Basic and general roles are role expressions whose syntax is given at point 3 below.
4. U denotes an *atomic attribute* (or simply attribute), and V a *general attribute*. An atomic attribute is an attribute denoted by a name, whereas a general attribute is an attribute expression whose syntax is given at point 4 below.

[4] `http://www.w3.org/RDF/`

We are now ready to define *DL-Lite*$_{\mathcal{A},id}$ expressions[5].

1. Concept expressions are built according to the following syntax:

$$\begin{array}{lcl} B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\ C & \longrightarrow & \top_c \mid B \mid \neg B \mid \exists Q.C \mid \delta_F(U) \end{array}$$

   Here, $\neg B$ denotes the *negation* of a basic concept B. The concept $\exists Q$, also called *unqualified existential restriction*, denotes the *domain* of a role Q, i.e., the set of objects that Q relates to some object. Similarly, $\delta(U)$ denotes the *domain* of an attribute U, i.e., the set of objects that U relates to some value. The concept $\exists Q.C$, also called *qualified existential restriction*, denotes the *qualified domain* of Q w.r.t. C, i.e., the set of objects that Q relates to some instance of C. Similarly, $\delta_F(U)$ denotes the *qualified domain* of U w.r.t. a value-domain F, i.e., the set of objects that U relates to some value in F.

2. Value-domain expressions are built according to the following syntax:

$$\begin{array}{lcl} E & \longrightarrow & \rho(U) \\ F & \longrightarrow & \top_d \mid T_1 \mid \cdots \mid T_n \end{array}$$

   Here, $\rho(U)$ denotes the *range* of an attribute U, i.e., the set of values to which U relates some object. Note that the range $\rho(U)$ of U is a value-domain, whereas the domain $\delta(U)$ of U is a concept.

3. Role expressions are built according to the following syntax:

$$\begin{array}{lcl} Q & \longrightarrow & P \mid P^- \\ R & \longrightarrow & Q \mid \neg Q \end{array}$$

   Here, P^- denotes the *inverse* of an atomic role, and $\neg Q$ denotes the *negation* of a basic role. In the following, when Q is a basic role, the expression Q^- stands for P^- when $Q = P$, and for P when $Q = P^-$.

4. Attribute expressions are built according to the following syntax:

$$V \longrightarrow U \mid \neg U$$

   Here, $\neg U$ denotes the *negation* of an atomic attribute.

As an example, consider the atomic concepts *Man* and *Woman*, and the atomic roles *HAS-HUSBAND*, representing the relationship between a woman and the man with whom she is married, and *HAS-CHILD*, representing the parent-child relationship. Then, intuitively, the inverse of *HAS-HUSBAND*, i.e., *HAS-HUSBAND*$^-$, represents the relationship between a man and his wife. Also, $\exists$*HAS-CHILD*.*Woman* represents those having a daughter.

[5] The results mentioned in this paper apply also to *DL-Lite*$_{\mathcal{A},id}$ extended with role attributes (cf. [19]), which are not considered here for the sake of simplicity.

$$\begin{array}{rcl}
A^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \\
(\exists Q)^{\mathcal{I}} & = & \{ o \mid \exists o'. (o,o') \in Q^{\mathcal{I}} \} \\
(\delta(U))^{\mathcal{I}} & = & \{ o \mid \exists v. (o,v) \in U^{\mathcal{I}} \} \\
(\exists Q.C)^{\mathcal{I}} & = & \{ o \mid \exists o'. (o,o') \in Q^{\mathcal{I}} \land o' \in C^{\mathcal{I}} \} \\
(\delta_F(U))^{\mathcal{I}} & = & \{ o \mid \exists v. (o,v) \in U^{\mathcal{I}} \land v \in F^{\mathcal{I}} \} \\
\top_c^{\mathcal{I}} & = & \Delta_O^{\mathcal{I}} \\
(\neg B)^{\mathcal{I}} & = & \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}}
\end{array}
\qquad
\begin{array}{rcl}
(\rho(U))^{\mathcal{I}} & = & \{ v \mid \exists o. (o,v) \in U^{\mathcal{I}} \} \\
\top_d^{\mathcal{I}} & = & \Delta_V^{\mathcal{I}} \\
T_i^{\mathcal{I}} & = & val(T_i) \\
P^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \\
(P^-)^{\mathcal{I}} & = & \{ (o,o') \mid (o',o) \in P^{\mathcal{I}} \} \\
(\neg Q)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}} \\
U^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}} \\
(\neg U)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}}
\end{array}$$

Fig. 1. Semantics of *DL-Lite*$_{\mathcal{A},id}$ expressions

Semantics. The meaning of *DL-Lite*$_{\mathcal{A},id}$ expressions is sanctioned by the semantics. Following the classical approach in DLs, the semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in terms of First-Order Logic interpretations. All such interpretations agree on the semantics assigned to each value-domain T_i and to each constant in Γ_V. In particular, each valued-domain T_i is interpreted as the set $val(T_i)$ of values of the corresponding RDF data type, and each constant $c_i \in \Gamma_V$ is interpreted as one specific value, denoted $val(c_i)$, in $val(T_i)$. Note that, since the data types T_i are pairwise disjoint, we have that $val(T_i) \cap val(T_j) = \emptyset$, for $i \neq j$.

Based on the above observations, we can now define the notion of interpretation in *DL-Lite*$_{\mathcal{A},id}$. An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where

- $\Delta^{\mathcal{I}}$ is the interpretation domain, which is the disjoint union of two non-empty sets: $\Delta_O^{\mathcal{I}}$, called the *domain of objects*, and $\Delta_V^{\mathcal{I}}$, called the *domain of values*. In turn, $\Delta_V^{\mathcal{I}}$ is the union of $val(T_1), \ldots, val(T_n)$.
- $\cdot^{\mathcal{I}}$ is the *interpretation function*, i.e., a function that assigns an element of $\Delta^{\mathcal{I}}$ to each constant in Γ, a subset of $\Delta^{\mathcal{I}}$ to each concept and value-domain, and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role and attribute, in such a way that the following holds:
  - for each $c \in \Gamma_V$, $c^{\mathcal{I}} = val(c)$,
  - for each $d \in \Gamma_O$, $d^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$,
  - for each $a_1, a_2 \in \Gamma$, $a_1 \neq a_2$ implies $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$, and
  - the conditions shown in Figure 1 are satisfied.

Note that the above definition implies that different constants are interpreted differently in the domain, i.e., *DL-Lite*$_{\mathcal{A},id}$ adopts the so-called *unique name assumption* (UNA).

2.2 *DL-Lite*$_{\mathcal{A},id}$ Ontologies

Like in any DL, a *DL-Lite*$_{\mathcal{A},id}$ *ontology*, or knowledge base (KB), is constituted by two components:

- a *TBox* (where the 'T' stands for *terminological*), which is a finite set of *intensional assertions*, and
- an *ABox* (where the 'A' stands for *assertional*), which is a finite set of *extensional* (or, *membership*) *assertions*.

We now specify formally the form of a *DL-Lite*$_{\mathcal{A},id}$ TBox and ABox, and its semantics.

Syntax. In *DL-Lite*$_{\mathcal{A},id}$, the TBox may contain intensional assertions of three types, namely inclusion assertions, functionality assertions, and local identification assertions.

– An *inclusion assertion* has one the forms

$$B \sqsubseteq C, \qquad E \sqsubseteq F, \qquad Q \sqsubseteq R, \qquad U \sqsubseteq V,$$

denoting respectively, from left to right, inclusions between concepts, value-domains, roles, and attributes. Intuitively, an inclusion assertion states that, in every model of $\mathcal{T}$, each instance of the left-hand side expression is also an instance of the right-hand side expression.

An inclusion assertion that on the right-hand side does not contain the symbol '$\neg$' is called a *positive inclusion (PI)*, while an inclusion assertion that on the right-hand side contains the symbol '$\neg$' is called a *negative inclusion (NI)*. Hence, a negative inclusion has one of the forms $B_1 \sqsubseteq \neg B_2$, $B_1 \sqsubseteq \exists Q_1.\exists Q_2.\ldots\exists Q_k.\neg B_2$, $P_1 \sqsubseteq \neg P_2$, or $U_1 \sqsubseteq \neg U_2$.

For example, the (positive) inclusion *Parent* $\sqsubseteq$ $\exists$*HAS-CHILD* specifies that parents have a child, the inclusions $\exists$*HAS-HUSBAND* $\sqsubseteq$ $Woman$ and $\exists$*HAS-HUSBAND*$^-$ $\sqsubseteq$ Man respectively specify that wifes (i.e., those who have a husband) are women and that husbands are men, and the inclusion *Person* $\sqsubseteq$ $\delta(\mathbf{hasSsn})$, where $\mathbf{hasSsn}$ is an attribute, specifies that each person has a social security number. The negative inclusion *Man* $\sqsubseteq$ $\neg$*Woman* specifies that men and women are disjoint.

– A *functionality assertion* has one of the forms

$$(\mathsf{funct}\ Q), \qquad (\mathsf{funct}\ U),$$

denoting functionality of a role and of an attribute, respectively. Intuitively, a functionality assertion states that the binary relation represented by a role (respectively, an attribute) is a function.

For example, the functionality assertion $(\mathsf{funct}$ *HAS-HUSBAND*$^-)$ states that a person may have at most one wife, and the functionality assertion $(\mathsf{funct}\ \mathbf{hasSsn})$ states that no individual may have more than one social security number.

– A *local identification assertion* (or, simply, identification assertion or identification constraint) makes use of the notion of path. A *path* is an expression built according to the following syntax,

$$\pi \longrightarrow S \mid D? \mid \pi \circ \pi \qquad (1)$$

where S denotes a basic role (i.e., an atomic role or the inverse of an atomic role), an atomic attribute, or the inverse of an atomic attribute, and $\pi_1 \circ \pi_2$ denotes the composition of the paths π_1 and π_2. Finally, D denotes an basic concept or a (basic or arbitrary) value domain, and the expression $D?$ is called a *test relation*, which represents the identity relation on instances of D. Test relations are used in all those cases in which one wants to impose that a path involves instances of a certain concept. For example, *HAS-CHILD* $\circ$ *Woman*? is the path connecting someone with his/her daughters.

A path π denotes a complex property for the instances of concepts: given an object o, every object that is reachable from o by means of π is called a π-*filler* for o. Note that for a certain o there may be several distinct π-fillers, or no π-fillers at all.
If π is a path, the length of π, denoted $\mathit{length}(\pi)$, is 0 if π has the form $D?$, is 1 if π has the form S, and is $\mathit{length}(\pi_1) + \mathit{length}(\pi_2)$ if π has the form $\pi_1 \circ \pi_2$. With the notion of path in place, we are ready for the definition of identification assertion, which is an assertion of the form

$$(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n),$$

where B is a basic concept, $n \geq 1$, and $\pi_1, \ldots, \pi_n$ (called the *components* of the identifier) are paths such that $\mathit{length}(\pi_i) \geq 1$ for all $i \in \{1, \ldots, n\}$. Intuitively, such a constraint asserts that for any two different instances o, o' of B, there is at least one π_i such that o and o' differ in the set of their π_i-fillers. The identification assertion is called *local* if $\mathit{length}(\pi_i) = 1$ for at least one $i \in \{1, \ldots, n\}$. The term "local" emphasizes that at least one of the paths has length 1 and thus refers to a local property of B. In the following, we will consider only local identification assertions, and thus simply omit the 'local' qualifier.

For example, the identification assertion $(\mathsf{id}$ *Woman HAS-HUSBAND*$)$ says that a woman is identified by her husband, i.e., there are not two different women with the same husband, whereas the identification assertion $(\mathsf{id}$ *Man HAS-CHILD*$)$ says that a man is identified by his children, i.e., there are not two men with a child in common. We can also say that there are not two men with the same daughters by means of the identification $(\mathsf{id}$ *Man HAS-CHILD* $\circ$ *Woman*?$)$.

Then, a *DL-Lite*$_{\mathcal{A},id}$ TBox is a finite sets of intensional assertions of the form above, where suitable limitations in the combination of such assertions are imposed. To precisely describe such limitations, we first introduce some preliminary notions. An atomic role P (resp., an atomic attribute U) is called an *identifying property in a TBox* $\mathcal{T}$, if

- $\mathcal{T}$ contains a functionality assertion $(\mathsf{funct}\ P)$ or $(\mathsf{funct}\ P^-)$ (resp., $(\mathsf{funct}\ U)$), or
- P (resp., U) appears (in either direct or inverse direction) in some path of an identification assertion in $\mathcal{T}$.

We say that an atomic role P (resp., an atomic attribute U) *appears positively* in the right-hand side of an inclusion assertion α if α has the form $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$, for some basic role Q (resp., $U' \sqsubseteq U$, for some atomic attribute U'). An atomic role P (resp., an atomic attribute U) is called *primitive in a TBox* $\mathcal{T}$, if

- it does not appear positively in the right-hand side of an inclusion assertion of $\mathcal{T}$, and
- it does not appear in $\mathcal{T}$ in an expression of the form $\exists P.C$ or $\exists P^-.C$ (resp., $\delta_F(U)$).

With these notions in place, we are ready to define what constitutes a *DL-Lite*$_{\mathcal{A},id}$ TBox.

Definition 2.1. *A DL-Lite$_{\mathcal{A},id}$ TBox, $\mathcal{T}$, is a finite set of inclusion assertions, functionality assertions, and identification assertions as specified above, and such that the following conditions are satisfied:*

(1) Each concept appearing in an identification assertion of $\mathcal{T}$ (either as the identified concept, or in some test relation of some path) is a basic concept, i.e., a concept of the form A, $\exists Q$, or $\delta(U)$.
(2) Each identifying property in $\mathcal{T}$ is primitive in $\mathcal{T}$.

A DL-Lite$_{\mathcal{A}}$ TBox is a DL-Lite$_{\mathcal{A},id}$ TBox that does not contain identification assertions.

Intuitively, the condition stated at point (2) says that, in *DL-Lite$_{\mathcal{A},id}$* TBoxes, roles and attributes occurring in functionality assertions or in paths of identification constraints cannot be specialized. We will see that the above conditions ensure the tractability of reasoning in our logic.

A *DL-Lite$_{\mathcal{A},id}$* (or *DL-Lite$_{\mathcal{A}}$*) ABox consists of a set of *membership assertions*, which are used to state the instances of concepts, roles, and attributes. Such assertions have the form

$$A(a), \qquad P(a_1, a_2), \qquad U(a, c),$$

where A is an atomic concept, P is an atomic role, U is an atomic attribute, a, a_1, a_2 are constants in Γ_O, and c is a constant in Γ_V.

Definition 2.2. *A DL-Lite$_{\mathcal{A},id}$ (resp., DL-Lite$_{\mathcal{A}}$) ontology $\mathcal{O}$ is a pair $\langle\mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ is a DL-Lite$_{\mathcal{A},id}$ (resp., DL-Lite$_{\mathcal{A}}$) TBox (cf. Definition 2.1), and $\mathcal{A}$ is a DL-Lite$_{\mathcal{A},id}$ (or DL-Lite$_{\mathcal{A}}$) ABox, all of whose atomic concepts, roles, and attributes occur in $\mathcal{T}$.*

Notice that, for an ontology $\mathcal{O} = \langle\mathcal{T}, \mathcal{A}\rangle$, the requirement in Definition 2.2 that all concepts, roles, and attributes that occur in $\mathcal{A}$ occur also in $\mathcal{T}$ is not a limitation. Indeed, as will be clear from the semantics, we can deal with the general case by adding to $\mathcal{T}$ inclusion assertions $A \sqsubseteq \top_c$, $\exists P \sqsubseteq \top_c$, and $\delta(U) \sqsubseteq \top_c$, for any atomic concepts A, atomic role P, and atomic attribute U occurring in $\mathcal{A}$ but not in $\mathcal{T}$, without altering the semantics of $\mathcal{O}$.

We also observe that in many DLs, functionality assertions are not explicitly present, since they can be expressed by means of number restrictions. *Number restrictions* $(\geq k\, Q)$ and $(\leq k\, Q)$, where k is a positive integer and Q a basic role, denotes the set of objects that are connected by means of role Q respectively to at least and at most k distinct objects. Hence, $(\geq k\, Q)$ generalizes existential quantification $\exists Q$, while $(\leq k\, Q)$ can be used to generalize functionality assertions. Indeed, the assertion $(\mathsf{funct}\ Q)$ is equivalent to the inclusion assertion $\exists Q \sqsubseteq (\leq 1\, Q)$, where the used number is 1, and the number restriction is expressed globally for the whole domain of Q. Instead, by means of an assertion $B \sqsubseteq (\leq k\, Q)$, one can impose *locally*, i.e., just for the instances of concept B, a numeric condition involving a number k that is different from 1.

Semantics. We now specify the semantics of an ontology, again in terms of interpretations, by defining when an interpretation $\mathcal{I}$ *satisfies* and assertion α (either an intensional assertion or a membership assertion), denoted $\mathcal{I} \models \alpha$.

– An interpretation $\mathcal{I}$ satisfies a concept (resp., value-domain, role, attribute) inclusion assertion

$$\begin{array}{lll} B \sqsubseteq C, & \text{if} & B^{\mathcal{I}} \subseteq C^{\mathcal{I}}; \\ E \sqsubseteq F, & \text{if} & E^{\mathcal{I}} \subseteq F^{\mathcal{I}}; \\ Q \sqsubseteq R, & \text{if} & Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}; \\ U \sqsubseteq V, & \text{if} & U^{\mathcal{I}} \subseteq V^{\mathcal{I}}. \end{array}$$

– An interpretation $\mathcal{I}$ satisfies a role functionality assertion $(\mathsf{funct}\ Q)$, if for each $o_1, o_2, o_3 \in \Delta_O^{\mathcal{I}}$

$$(o_1, o_2) \in Q^{\mathcal{I}} \text{ and } (o_1, o_3) \in Q^{\mathcal{I}} \quad \text{implies} \quad o_2 = o_3.$$

– An interpretation $\mathcal{I}$ satisfies an attribute functionality assertion $(\mathsf{funct}\ U)$, if for each $o \in \Delta_O^{\mathcal{I}}$ and $v_1, v_2 \in \Delta_V^{\mathcal{I}}$

$$(o, v_1) \in U^{\mathcal{I}} \text{ and } (o, v_2) \in U^{\mathcal{I}} \quad \text{implies} \quad v_1 = v_2.$$

– In order to define the semantics of identification assertions, we first define the semantics of paths. The extension $\pi^{\mathcal{I}}$ of a path π in an interpretation $\mathcal{I}$ is defined as follows:
  - if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
  - if $\pi = D?$, then $\pi^{\mathcal{I}} = \{(o, o) \mid o \in D^{\mathcal{I}}\}$,
  - if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

  As a notation, we write $\pi^{\mathcal{I}}(o)$ to denote the set of π-fillers for o in $\mathcal{I}$, i.e., $\pi^{\mathcal{I}}(o) = \{o' \mid (o, o') \in \pi^{\mathcal{I}}\}$.

  Then, an interpretation $\mathcal{I}$ satisfies an identification assertion $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$ if for all $o, o' \in B^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Observe that this definition is coherent with the intuitive reading of identification assertions discussed above, in particular by sanctioning that two different instances o, o' of B differ in the set of their π_i-fillers when such sets are disjoint.[6]

– An interpretation $\mathcal{I}$ satisfies a membership assertion

$$\begin{array}{lll} A(a), & \text{if} & a^{\mathcal{I}} \in A^{\mathcal{I}}; \\ P(a_1, a_2), & \text{if} & (a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}; \\ U(a, c), & \text{if} & (a^{\mathcal{I}}, c^{\mathcal{I}}) \in U^{\mathcal{I}}. \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$, ABox $\mathcal{A}$), or, equivalently, $\mathcal{I}$ *satisfies* $\mathcal{O}$ (resp., $\mathcal{T}$, $\mathcal{A}$), written $\mathcal{I} \models \mathcal{O}$ (resp., $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$) if and only if $\mathcal{I}$ satisfies all assertions in $\mathcal{O}$ (resp., $\mathcal{T}$, $\mathcal{A}$). The semantics of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is the set of all *models* of $\mathcal{O}$.

[6] Note that an alternative definition of the semantics of identification assertions is the one where an interpretation $\mathcal{I}$ satisfies $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$ if for all $o, o' \in B^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) = \pi_1^{\mathcal{I}}(o') \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) = \pi_n^{\mathcal{I}}(o')$ implies $o = o'$. This alternative semantics coincides with the one we have adopted in the case where all roles and attributes in all paths π_i are functional, but imposes a stronger condition for identification when this is not the case. Indeed, the alternative semantics sanctions that two different instances o, o' of B differ in the set of their π_i-fillers when such sets are *different* (rather than disjoint).

With the semantics of an ontology in place, we comment briefly on the various types of assertions in a *DL-Lite*$_{\mathcal{A},id}$ TBox and relate them to constraints used in classical database theory. We remark, however, that TBox assertions have a fundamentally different role from the one of database dependencies: while the latter are typically enforced on the data in a database, this is not the case for the former, which are instead used to infer new knowledge from the asserted one.

– Inclusion assertions having a positive element in the right-hand side intuitively correspond to inclusion dependencies in databases [1]. Specifically Concept inclusions correspond to unary inclusion dependencies [34], while role inclusions correspond to binary inclusion dependencies. An inclusion assertion of the form $\exists Q \sqsubseteq A$ is in fact a *foreign key*, since objects that are instances of concept A can be considered as keys for the (unary) relations denoted by A. Instead, an inclusion of the form $A \sqsubseteq \exists Q$ can be considered as a participation constraint.
– Inclusion assertions having a negative element in the right-hand side intuitively correspond to exclusion (or disjointness) dependencies in databases [1].
– Functionality assertions correspond to unary key dependencies in databases. Specifically $(\mathsf{funct}\ P)$ corresponds to stating that the first component of the binary relation P is a key for P, while $(\mathsf{funct}\ P^-)$ states the same for the second component of P.
– Identification assertions correspond to more complex forms of key dependencies. To illustrate this correspondence, consider a concept A and a set of attributes $U_1, \ldots, U_n$, where each U_i is functional and has A as domain (i.e., the TBox contains the functionality assertion $(\mathsf{funct}\ U_i)$ and the inclusion assertion $\delta(U_i) \sqsubseteq A$). Together, A and "its attributes" can be considered as representing a single relation R_A of arity $n+1$ constituted by one column for the object A and one column for each of the U_i attributes. Then, an identification assertion $(\mathsf{id}\ A\ U_{i_1}, \ldots, U_{i_k})$, involving a subset $U_{i_1}, \ldots, U_{i_k}$ of the attributes of A, resembles a key dependency on R_A, where the key is given by the specified subset of attributes. Indeed, due to the identification assertion, a given sequence $v_1, \ldots, v_k$ of values for $U_{i_1}, \ldots, U_{i_k}$ determines a unique instance a of A, and since all attributes are functional and have A as domain, their value is uniquely determined by a, and hence by $v_1, \ldots, v_k$.

For further intuitions about the meaning of the various kinds of TBox assertions we refer also to Section 3, where the relationship with conceptual models (specifically, UML class diagrams) is discussed in detail.

Example 2.3. We conclude this section with an example in which we present a *DL-Lite*$_{\mathcal{A},id}$ ontology modeling the annual national football[7] championships in Europe, where the championship for a specific nation is called *league* (e.g., the Spanish Liga). A league is structured in terms of a set of *rounds*. Every round contains a set of *matches*, each one characterized by one *home team* and one *host team*. We distinguish between scheduled matches, i.e., matches that have still to be played, and played matches. Obviously, a match falls in exactly one of these two categories.

[7] Football is called "soccer" in the United States.

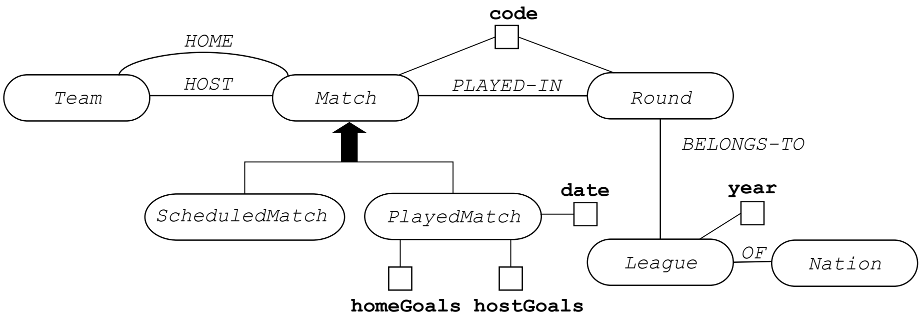


Fig. 2. Diagrammatic representation of the football championship ontology

Inclusion assertions					
League	$\sqsubseteq$	$\exists \mathit{OF}$	$\mathit{PlayedMatch}$	$\sqsubseteq$	Match
$\exists \mathit{OF}$	$\sqsubseteq$	League	$\mathit{ScheduledMatch}$	$\sqsubseteq$	Match
$\exists \mathit{OF}^-$	$\sqsubseteq$	Nation	$\mathit{PlayedMatch}$	$\sqsubseteq$	$\neg \mathit{ScheduledMatch}$
Round	$\sqsubseteq$	$\exists \mathit{BELONGS\text{-}TO}$	Match	$\sqsubseteq$	$\neg \mathit{Round}$
$\exists \mathit{BELONGS\text{-}TO}$	$\sqsubseteq$	Round	League	$\sqsubseteq$	$\delta(\mathbf{year})$
$\exists \mathit{BELONGS\text{-}TO}^-$	$\sqsubseteq$	League	Match	$\sqsubseteq$	$\delta(\mathbf{code})$
Match	$\sqsubseteq$	$\exists \mathit{PLAYED\text{-}IN}$	Round	$\sqsubseteq$	$\delta(\mathbf{code})$
$\exists \mathit{PLAYED\text{-}IN}$	$\sqsubseteq$	Match	$\mathit{PlayedMatch}$	$\sqsubseteq$	$\delta(\mathbf{date})$
$\exists \mathit{PLAYED\text{-}IN}^-$	$\sqsubseteq$	Round	$\mathit{PlayedMatch}$	$\sqsubseteq$	$\delta(\mathbf{homeGoals})$
Match	$\sqsubseteq$	$\exists \mathit{HOME}$	$\mathit{PlayedMatch}$	$\sqsubseteq$	$\delta(\mathbf{hostGoals})$
$\exists \mathit{HOME}$	$\sqsubseteq$	Match	$\rho(\mathbf{date})$	$\sqsubseteq$	`xsd:date`
$\exists \mathit{HOME}^-$	$\sqsubseteq$	Team	$\rho(\mathbf{homeGoals})$	$\sqsubseteq$	`xsd:nonNegativeInteger`
Match	$\sqsubseteq$	$\exists \mathit{HOST}$	$\rho(\mathbf{hostGoals})$	$\sqsubseteq$	`xsd:nonNegativeInteger`
$\exists \mathit{HOST}$	$\sqsubseteq$	Match	$\rho(\mathbf{code})$	$\sqsubseteq$	`xsd:string`
$\exists \mathit{HOST}^-$	$\sqsubseteq$	Team	$\rho(\mathbf{year})$	$\sqsubseteq$	`xsd:positiveInteger`

Functionality assertions			
(funct *OF*)	(funct *HOME*)	(funct **year**)	(funct **homeGoals**)
(funct *BELONGS-TO*)	(funct *HOST*)	(funct **code**)	(funct **hostGoals**)
(funct *PLAYED-IN*)		(funct **date**)	

Identification assertions	
1. (id *League* *OF*, **year**)	6. (id *PlayedMatch* **date**, *HOST*)
2. (id *Round* *BELONGS-TO*, **code**)	7. (id *PlayedMatch* **date**, *HOME*)
3. (id *Match* *PLAYED-IN*, **code**)	8. (id *League* **year**, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOME}$)
4. (id *Match* *HOME*, *PLAYED-IN*)	9. (id *League* **year**, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOST}$)
5. (id *Match* *HOST*, *PLAYED-IN*)	10. (id *Match* *HOME*, *HOST*, $\mathit{PLAYED\text{-}IN} \circ \mathit{BELONGS\text{-}TO} \circ \mathbf{year}$)

Fig. 3. The $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ for the football championship example

In Figure 2, we show a schematic representation of (a portion of) the intensional part of the ontology for the football championship domain. In this figure, the black arrow represents a partition of one concept into a set of sub-concepts. We have not represented explicitly in the figure the pairwise disjointness of the concepts *Team*, *Match*, *Round*, *League*, and *Nation*, which intuitively holds in the modeled domain. In Figure 3, a $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ is shown that captures (most of) the above aspects. In our examples, we use the *CapitalizedItalics* font to denote atomic concepts, the *ALL-CAPITALS-ITALICS* font to denote atomic roles, the `typewriter` font to denote value-domains, and the **boldface** font to denote atomic attributes. Regarding the pairwise disjointness of the various concepts, we have represented by means of negative inclusion assertions only the disjointness between *PlayedMatch* and *ScheduledMatch* and the one between *Match* and *Round*. By virtue of the characteristics of *DL-Lite*$_{\mathcal{A},id}$, we can explicitly consider also attributes of concepts and the fact that they are used for identification. In particular, we assume that when a scheduled match takes place, it is played in a specific date, and that for every match that has been played, the number of goals scored by the home team and by the host team are given. Note that different matches scheduled for the same round can be played in different dates. Also, we want to distinguish football championships on the basis of the nation and the year in which a championship takes place (e.g., the 2009 Italian Liga). We also assume that both matches and rounds have codes. The identification assertions model the following aspects:

1. No nation has two leagues in the same year.
2. Within a league, the code associated to a round is unique.
3. Every match is identified by its code within its round.
4. A team is the home team of at most one match per round.
5. As above for the host team.
6. No home team participates in different played matches in the same date.
7. As above for the host team.
8. No home team plays in different leagues in the same year.
9. As above for the host team.
10. No pair (home team, host team) plays different matches in the same year.

Note that the *DL-Lite*$_{\mathcal{A},id}$ TBox in Figure 3 captures the ontology in Figure 2, except for the fact that the concept *Match* covers the concepts *ScheduledMatch* and *PlayedMatch*. In order to express such a condition, we would need to use disjunction in the right-hand-side of inclusion assertions, i.e.,

$$\mathit{Match} \sqsubseteq \mathit{ScheduledMatch} \sqcup \mathit{PlayedMatch}$$

where $\sqcup$ would be interpreted as set union. As we will see in Section 6, we have to renounce to the expressive power required to capture covering constraints (i.e., disjunction), if we want to preserve nice computational properties for reasoning over *DL-Lite*$_{\mathcal{A},id}$ ontologies.

An ABox, $\mathcal{A}_{\mathit{fbc}}$, associated to the TBox in Figure 3 is shown in Figure 4, where we have used the *slanted* font for constants in Γ_O and the `typeface` font for constants in Γ_V. For convenience of reading, we have chosen in the example names of the constants that indicate the properties of the objects that the constants represent.

We observe that the ontology $\mathcal{O}_{\mathit{fbc}} = \langle \mathcal{T}_{\mathit{fbc}}, \mathcal{A}_{\mathit{fbc}} \rangle$ is satisfiable. Indeed, the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ shown in Figure 5 is a model of the ABox $\mathcal{A}_{\mathit{fbc}}$, where we have assumed that for each value constant $c \in \Gamma_V$, the corresponding value $val(c)$ is equal to c itself, hence $c^{\mathcal{I}} = val(c) = c$. Moreover, it is easy to see that every interpretation $\mathcal{I}$ has to satisfy the conditions shown in Figure 5 in order to be a model of $\mathcal{A}_{\mathit{fbc}}$.

CONCEPT AND ROLE MEMBERSHIP ASSERTIONS		
$\mathit{League}(\mathit{it2009})$		$\mathit{PLAYED\text{-}IN}(\mathit{m7RJ}, \mathit{r7})$
$\mathit{Round}(\mathit{r7})$	$\mathit{BELONGS\text{-}TO}(\mathit{r7}, \mathit{it2009})$	$\mathit{PLAYED\text{-}IN}(\mathit{m8NT}, \mathit{r8})$
$\mathit{Round}(\mathit{r8})$	$\mathit{BELONGS\text{-}TO}(\mathit{r8}, \mathit{it2009})$	$\mathit{PLAYED\text{-}IN}(\mathit{m8RM}, \mathit{r8})$
$\mathit{PlayedMatch}(\mathit{m7RJ})$	$\mathit{HOME}(\mathit{m7RJ}, \mathit{roma})$	$\mathit{HOST}(\mathit{m7RJ}, \mathit{juventus})$
$\mathit{Match}(\mathit{m8NT})$	$\mathit{HOME}(\mathit{m8NT}, \mathit{napoli})$	$\mathit{HOST}(\mathit{m8NT}, \mathit{torino})$
$\mathit{Match}(\mathit{m8RM})$	$\mathit{HOME}(\mathit{m8RM}, \mathit{roma})$	$\mathit{HOST}(\mathit{m8RM}, \mathit{milan})$
$\mathit{Team}(\mathit{roma})$	$\mathit{Team}(\mathit{napoli})$	$\mathit{Team}(\mathit{juventus})$
ATTRIBUTE MEMBERSHIP ASSERTIONS		
$\mathbf{code}(\mathit{r7}, \texttt{"7"})$	$\mathbf{code}(\mathit{m7RJ}, \texttt{"RJ"})$	$\mathbf{date}(\mathit{m7RJ}, \texttt{5/4/09})$
$\mathbf{code}(\mathit{r8}, \texttt{"8"})$	$\mathbf{code}(\mathit{m8NT}, \texttt{"NT"})$	$\mathbf{homeGoals}(\mathit{m7RJ}, \texttt{3})$
	$\mathbf{code}(\mathit{m8RM}, \texttt{"RM"})$	$\mathbf{hostGoals}(\mathit{m7RJ}, \texttt{1})$

Fig. 4. The ABox $\mathcal{A}_{\mathit{fbc}}$ for the football championship example

$(\mathit{it2009}^{\mathcal{I}}) \in \mathit{League}^{\mathcal{I}}$		$(\mathit{m7RJ}^{\mathcal{I}}, \mathit{r7}^{\mathcal{I}}) \in \mathit{PLAYED\text{-}IN}^{\mathcal{I}}$
$(\mathit{r7}^{\mathcal{I}}) \in \mathit{Round}^{\mathcal{I}}$	$(\mathit{r7}^{\mathcal{I}}, \mathit{it2009}^{\mathcal{I}}) \in \mathit{BELONGS\text{-}TO}^{\mathcal{I}}$	$(\mathit{m8NT}^{\mathcal{I}}, \mathit{r8}^{\mathcal{I}}) \in \mathit{PLAYED\text{-}IN}^{\mathcal{I}}$
$(\mathit{r8}^{\mathcal{I}}) \in \mathit{Round}^{\mathcal{I}}$	$(\mathit{r8}^{\mathcal{I}}, \mathit{it2009}^{\mathcal{I}}) \in \mathit{BELONGS\text{-}TO}^{\mathcal{I}}$	$(\mathit{m8RM}^{\mathcal{I}}, \mathit{r8}^{\mathcal{I}}) \in \mathit{PLAYED\text{-}IN}^{\mathcal{I}}$
$(\mathit{m7RJ}^{\mathcal{I}}) \in \mathit{PlayedMatch}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \mathit{roma}^{\mathcal{I}}) \in \mathit{HOME}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \mathit{juventus}^{\mathcal{I}}) \in \mathit{HOST}^{\mathcal{I}}$
$(\mathit{m8NT}^{\mathcal{I}}) \in \mathit{Match}^{\mathcal{I}}$	$(\mathit{m8NT}^{\mathcal{I}}, \mathit{napoli}^{\mathcal{I}}) \in \mathit{HOME}^{\mathcal{I}}$	$(\mathit{m8NT}^{\mathcal{I}}, \mathit{torino}^{\mathcal{I}}) \in \mathit{HOST}^{\mathcal{I}}$
$(\mathit{m8RM}^{\mathcal{I}}) \in \mathit{Match}^{\mathcal{I}}$	$(\mathit{m8RM}^{\mathcal{I}}, \mathit{roma}^{\mathcal{I}}) \in \mathit{HOME}^{\mathcal{I}}$	$(\mathit{m8RM}^{\mathcal{I}}, \mathit{milan}^{\mathcal{I}}) \in \mathit{HOST}^{\mathcal{I}}$
$(\mathit{roma}^{\mathcal{I}}) \in \mathit{Team}^{\mathcal{I}}$	$(\mathit{napoli}^{\mathcal{I}}) \in \mathit{Team}^{\mathcal{I}}$	$(\mathit{juventus}^{\mathcal{I}}) \in \mathit{Team}^{\mathcal{I}}$
$(\mathit{r7}^{\mathcal{I}}, \texttt{"7"}) \in \mathbf{code}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \texttt{"RJ"}) \in \mathbf{code}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \texttt{5/4/09}) \in \mathbf{date}^{\mathcal{I}}$
$(\mathit{r8}^{\mathcal{I}}, \texttt{"8"}) \in \mathbf{code}^{\mathcal{I}}$	$(\mathit{m8NT}^{\mathcal{I}}, \texttt{"NT"}) \in \mathbf{code}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \texttt{3}) \in \mathbf{homeGoals}^{\mathcal{I}}$
	$(\mathit{m8RM}^{\mathcal{I}}, \texttt{"RM"}) \in \mathbf{code}^{\mathcal{I}}$	$(\mathit{m7RJ}^{\mathcal{I}}, \texttt{1}) \in \mathbf{hostGoals}^{\mathcal{I}}$

Fig. 5. A model of the ABox $\mathcal{A}_{\mathit{fbc}}$ for the football championship example

Furthermore, the following are necessary conditions for $\mathcal{I}$ to be also a model of the TBox $\mathcal{T}_{\mathit{fbc}}$, and hence of $\mathcal{O}_{\mathit{fbc}}$:

$$\begin{array}{lll}
\mathit{it2009}^{\mathcal{I}} \in (\exists \mathit{OF})^{\mathcal{I}} & \text{to satisfy} & \mathit{League} \sqsubseteq \exists \mathit{OF}, \\
\mathit{it2009}^{\mathcal{I}} \in (\delta(\mathbf{year}))^{\mathcal{I}} & \text{to satisfy} & \mathit{League} \sqsubseteq \delta(\mathbf{year}), \\
\mathit{m7RJ}^{\mathcal{I}} \in \mathit{Match}^{\mathcal{I}} & \text{to satisfy} & \mathit{PlayedMatch} \sqsubseteq \mathit{Match}, \\
\mathit{torino}^{\mathcal{I}} \in \mathit{Team}^{\mathcal{I}} & \text{to satisfy} & \exists \mathit{HOST}^{-} \sqsubseteq \mathit{Team}, \\
\mathit{milan}^{\mathcal{I}} \in \mathit{Team}^{\mathcal{I}} & \text{to satisfy} & \exists \mathit{HOST}^{-} \sqsubseteq \mathit{Team}.
\end{array}$$

Notice that, in order for an interpretation $\mathcal{I}$ to satisfy the condition specified in the first row above, there must be an object $o \in \Delta_O^{\mathcal{I}}$ such that $(\mathit{it2009}^{\mathcal{I}}, o) \in \mathit{OF}^{\mathcal{I}}$. According to the inclusion assertion $\exists \mathit{OF}^{-} \sqsubseteq \mathit{Nation}$, such an object o must also belong to $\mathit{Nation}^{\mathcal{I}}$ (indeed, in our ontology, every league is of one nation). Similarly, the second row above derives from the property that every league must have a year.

We note that, besides satisfying the conditions discussed above, an interpretation $\mathcal{I}'$ may also add other elements to the interpretation of concepts, attributes, or roles specified by $\mathcal{I}$. For instance, the interpretation $\mathcal{I}'$ that adds to $\mathcal{I}$ the object

$$italy^{\mathcal{I}} \in Nation^{\mathcal{I}}$$

is still a model of the ontology $\mathcal{O}_{fbc}$.

Note, finally, that there exists no model of $\mathcal{O}_{fbc}$ such that *m7RJ* is interpreted as an instance of *ScheduledMatch*, since *m7RJ* has to be interpreted as an instance of *PlayedMatch*, and according to the inclusion assertion

$$PlayedMatch \sqsubseteq \neg ScheduledMatch,$$

the sets of played matches and of scheduled matches are disjoint. ■

The above example clearly shows the difference between a database and an ontology. From a database point of view the ontology $\mathcal{O}_{fbc}$ discussed in the example might seem incorrect: for example, while the TBox $\mathcal{T}_{fbc}$ sanctions that every league has a year, there is no explicit year for *it2009* in the ABox $\mathcal{A}_{fbc}$. However, the ontology is not incorrect: the axiom stating that every league has a year simply specifies that in every model of $\mathcal{O}_{fbc}$ there will be a year for *it2009*, even if such a year is not known.

2.3 *DL-Lite*$_{\mathcal{A},id}$ vs. OWL 2 QL

Having now completed the definition of the syntax and semantics of *DL-Lite*$_{\mathcal{A},id}$, we would like to point out that *DL-Lite*$_{\mathcal{A},id}$ is at the basis of OWL 2 QL, one of the three profiles of OWL 2 that are currently being standardized by the World-Wide-Web Consortium (W3C). The *OWL 2 profiles*[8] are fragments of the full OWL 2 language that have been designed and standardized for specific application requirements. According to (the current version of) the official W3C profiles document, "OWL 2 QL includes most of the main features of conceptual models such as UML class diagrams and ER diagrams. [It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems." We will substantiate all these claims in the next sections.

Here, we briefly point out the most important differences between *DL-Lite*$_{\mathcal{A},id}$ and OWL 2 QL (apart from differences in terminology and syntax, which we do not mention):

(1) The main difference is certainly the fact that OWL 2 QL does not adopt the *unique name assumption*, while such assumption holds for *DL-Lite*$_{\mathcal{A},id}$ (and the whole *DL-Lite* family, in fact). The reason for this semantic mismatch is on the one hand that OWL 2, as most DLs, does not adopt the UNA, and since the profiles are intended to be *syntactic* fragments of the full OWL 2 language, it was not desirable for a profile to change a basic semantic assumption. On the other hand, the UNA is at the basis of data management in databases, and moreover, by dropping it, *DL-Lite*$_{\mathcal{A},id}$ would lose its nice computational properties (cf. Theorem 6.6).

[8] `http://www.w3.org/TR/owl2-profiles/`

(2) OWL 2 QL does not allow for expressing *functionality* of roles or attributes, or *identification assertions*, while such constructs are present in *DL-Lite*$_{\mathcal{A},id}$. This aspect is related to Item (1), and motivated by the fact that the OWL 2 QL profile is intended to have the same nice computational properties as *DL-Lite*$_{\mathcal{A},id}$. In order to preserve such properties even in the absence of the UNA, the proof of Theorem 6.6 tells us that we need to avoid the use of functionality (and of identification assertions, since these can be used to simulate functionality). Indeed, as testified also by the complexity results in [4], in the absence of these constructs, the UNA has no impact on complexity of reasoning, and hence OWL 2 QL exhibits the same computational properties as *DL-Lite*$_{\mathcal{A},id}$.

(3) OWL 2 QL includes the possibility to assert additional role properties, such as disjointness, reflexivity, irreflexivity, symmetry, and asymmetry, that are not explicitly present in *DL-Lite*$_{\mathcal{A},id}$. It is immediate to see that disjointness between roles Q_1 and Q_2 can be expressed by means of $Q_1 \sqsubseteq \neg Q_2$, and that reflexivity of a role P can be expressed by means of $P \sqsubseteq P^-$. Moreover, as shown in [4], also the addition of irreflexivity, symmetry, and asymmetry does not affect the computational complexity of inference (including query answering, see Section 2.4), and such constructs could be incorporated in the reasoning algorithms for *DL-Lite*$_{\mathcal{A},id}$ with only minor changes.

(4) OWL 2 QL inherits its specific *datatypes* (corresponding to the value domains of *DL-Lite*$_{\mathcal{A},id}$) from OWL 2, while *DL-Lite*$_{\mathcal{A},id}$ does not provide any details about datatypes. However, OWL 2 QL imposes restrictions on the allowed datatypes that ensure that no datatype has an unbounded domain, which is sufficient to guarantee that datatypes will not interfere unexpectedly in reasoning.

We remark that, due to the correspondence between OWL 2 QL and *DL-Lite*$_{\mathcal{A},id}$, all the results and techniques presented in the next sections have a direct impact on OWL 2, i.e., on a standard language for the Semantic Web, that builds on a large user base. Hence, such results are of immediate practical relevance.

2.4 Queries over *DL-Lite*$_{\mathcal{A},id}$ Ontologies

We are interested in queries over ontologies expressed in *DL-Lite*$_{\mathcal{A},id}$. Similarly to the case of relational databases, the basic query class that we consider is the class of unions of conjunctive queries, which is a subclass of the class of First-Order Logic queries.

Syntax of Queries. A First-Order Logic (FOL) *query* q over a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is a, possibly open, FOL formula $\varphi(\boldsymbol{x})$ whose predicate symbols are atomic concepts, value-domains, roles, or attributes of $\mathcal{O}$ (resp., $\mathcal{T}$). The free variables of $\varphi(\boldsymbol{x})$ are those appearing in $\boldsymbol{x}$, which is a tuple of (pairwise distinct) variables. In other words, the atoms of $\varphi(\boldsymbol{x})$ have the form $A(x)$, $D(x)$, $P(x, y)$, $U(x, y)$, or $x = y$, where:

– A, F, P, and U are respectively an atomic concept, a value-domain, an atomic role, and an atomic attribute in $\mathcal{O}$,
– x, y are either variables in $\boldsymbol{x}$ or constants in Γ.

The *arity* of q is the arity of $\boldsymbol{x}$. A query of arity 0 is called a *boolean query*. When we want to make the arity of a query q explicit, we denote the query as $q(\boldsymbol{x})$.

A *conjunctive query* (CQ) $q(\boldsymbol{x})$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology is a FOL query of the form

$$\exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y}),$$

where $\boldsymbol{y}$ is a tuple of pairwise distinct variables not occurring among the free variables $\boldsymbol{x}$, and where $conj(\boldsymbol{x}, \boldsymbol{y})$ is a *conjunction* of atoms. The variables $\boldsymbol{x}$ are also called *distinguished* and the (existentially quantified) variables $\boldsymbol{y}$ are called *non-distinguished*. We will also make use of *conjunctive queries with inequalities*, which are CQs in which also atoms of the form $x \neq y$ (called *inequalities*) may appear.

A *union of conjunctive queries* (UCQ) is a FOL query that is the disjunction of a set of CQs of the same arity, i.e., it is a FOL formula of the form:

$$\exists \boldsymbol{y}_1.\, conj_1(\boldsymbol{x}, \boldsymbol{y}_1) \vee \cdots \vee \exists \boldsymbol{y}_n.\, conj_n(\boldsymbol{x}, \boldsymbol{y}_n).$$

UCQs with inequalities are obvious extensions of UCQs.

Finally, a *positive FOL query* is a FOL query $\varphi(\boldsymbol{x})$ where the formula φ is built using only conjunction, disjunction, and existential quantification (i.e., it contains neither negation nor universal quantification).

Datalog Notation for CQs and UCQs. In the following, it will sometimes be convenient to consider a UCQ as a set of CQs, rather than as a disjunction of UCQs. We will also use the *Datalog* notation for CQs and UCQs. In this notation, a CQ is written as

$$q(\boldsymbol{x}) \leftarrow conj'(\boldsymbol{x}, \boldsymbol{y})$$

and a UCQ is written as a set of CQs

$$\begin{array}{c} q(\boldsymbol{x}) \leftarrow conj'_1(\boldsymbol{x}, \boldsymbol{y}_1) \\ \vdots \\ q(\boldsymbol{x}) \leftarrow conj'_n(\boldsymbol{x}, \boldsymbol{y}_n) \end{array}$$

where $conj'(\boldsymbol{x}, \boldsymbol{y})$ and each $conj'_i(\boldsymbol{x}, \boldsymbol{y}_i)$ in a CQ are considered simply as sets of atoms (written in list notation, using a ',' as a separator). In this case, we say that $q(\boldsymbol{x})$ is the *head* of the query, and that $conj'(\boldsymbol{x}, \boldsymbol{y})$ and each $conj'_i(\boldsymbol{x}, \boldsymbol{y}_i)$ is the *body* of the corresponding CQ.

Semantics of Queries. Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, the FOL query $q = \varphi(\boldsymbol{x})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\boldsymbol{o} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that the formula φ evaluates to true in $\mathcal{I}$ under the assignment that assigns each object in $\boldsymbol{o}$ to the corresponding variable in $\boldsymbol{x}$ [1]. We call $q^{\mathcal{I}}$ the *answer* to q over $\mathcal{I}$. Notice that the answer to a boolean query is either the empty tuple, "()", considered as $true$, or the empty set, considered as $false$.

We remark that a relational database (over the atomic concepts, roles, and attributes) corresponds to a finite interpretation. Hence the notion of answer to a query introduced here is the standard notion of answer to a query evaluated over a relational database.

In the case where the query is a CQ, the above definition of answer can be rephrased in terms of homomorphisms. In general, a homomorphisms between two interpretations (i.e., First-Order structures) is defined as follows.

Definition 2.4. *Given two interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ over the same set $\mathcal{P}$ of predicate symbols, a* homomorphism *μ from $\mathcal{I}$ to $\mathcal{J}$ is a mapping $\mu : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ such that, for each predicate $P \in \mathcal{P}$ of arity n and each tuple $(o_1, \ldots, o_n) \in (\Delta^{\mathcal{I}})^n$, if $(o_1, \ldots, o_n) \in P^{\mathcal{I}}$, then $(\mu(o_1), \ldots, \mu(o_n)) \in P^{\mathcal{J}}$.*

Notice that, in the case of interpretations of a *DL-Lite*$_{\mathcal{A},id}$ ontology, the set of predicate symbols in the above definition would be the set of atomic concepts, value domains, roles, and attributes of the ontology.

We can now extend the definition to consider also homomorphisms from CQs to interpretations.

Definition 2.5. *Given a CQ $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$ over interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and a tuple $\boldsymbol{o} = (o_1, \ldots, o_n)$ of objects of $\Delta^{\mathcal{I}}$ of the same arity as $\boldsymbol{x} = (x_1, \ldots, x_n)$, a* homomorphism *from $q(\boldsymbol{o})$ to $\mathcal{I}$ is a mapping μ from the variables and constants in $q(\boldsymbol{x})$ to $\Delta^{\mathcal{I}}$ such that:*

- *$\mu(c) = c^{\mathcal{I}}$, for each constant c in $conj(\boldsymbol{x}, \boldsymbol{y})$,*
- *$\mu(x_i) = o_i$, for $i \in \{1, \ldots, n\}$, and*
- *$(\mu(t_1), \ldots, \mu(t_n)) \in P^{\mathcal{I}}$, for each atom $P(t_1, \ldots, t_n)$ that appears in $conj(\boldsymbol{x}, \boldsymbol{y})$.*

The following result established in [32] provides a fundamental characterization of answers to CQs in terms of homomorphism.

Theorem 2.6 ([32]). *Given a CQ $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$ over an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and a tuple $\boldsymbol{o} = (o_1, \ldots, o_n)$ of objects of $\Delta^{\mathcal{I}}$ of the same arity as $\boldsymbol{x} = (x_1, \ldots, x_n)$, we have that $\boldsymbol{o} \in q^{\mathcal{I}}$ if and only if there is a homomorphism from $q(\boldsymbol{o})$ to $\mathcal{I}$.*

In fact, the notion of homomorphism is crucial in the context of the study of CQs, and most inference tasks involving CQs (including query containment [58], and tasks related to view-based query processing [46]) can be rephrased in terms of homomorphism [1].

Example 2.7. Consider again the ontology $\mathcal{O}_{fbc} = \langle \mathcal{T}_{fbc}, \mathcal{A}_{fbc} \rangle$ introduced in Example 2.3, and the following query asking for all matches:

$$q_1(x) \leftarrow Match(x).$$

If $\mathcal{I}$ is the interpretation shown in Figure 5, we have that:

$$q_1^{\mathcal{I}} = \{(m8NT^{\mathcal{I}}), (m8RM^{\mathcal{I}})\}.$$

Notice that $\mathcal{I}$ is a model of $\mathcal{A}_{fbc}$, but not of $\mathcal{T}_{fbc}$. Let instead $\mathcal{I}'$ be the interpretation analogous to $\mathcal{I}$, but extended in such a way that it becomes also a model of $\mathcal{T}_{fbc}$, and hence of $\mathcal{O}_{fbc}$, as shown in Example 2.3. Then we have that:

$$q_1^{\mathcal{I}'} = \{(m8NT^{\mathcal{I}}),\ (m8RM^{\mathcal{I}}),\ (m7RJ^{\mathcal{I}})\}.$$

Suppose now that we ask for teams, together with the code of the match in which they have played as home team:

$$q_2(t, c) \leftarrow \mathit{Team}(t), \mathit{HOME}(m, t), \mathit{Match}(m), \mathbf{code}(m, c).$$

Then we have that

$$\begin{aligned} q_2^{\mathcal{I}} &= \{(\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\}, \\ q_2^{\mathcal{I}'} &= \{(\mathit{roma}^{\mathcal{I}}, \texttt{"RJ"}),\ (\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\}. \end{aligned}$$

■

Certain Answers. The notion of answer to a query introduced above is not sufficient to capture the situation where a query is posed over an ontology, since in general an ontology will have many models, and we cannot single out a unique interpretation (or database) over which to answer the query. Instead, the ontology determines a set of interpretations, i.e., the set of its models, which intuitively can be considered as the set of databases that are "compatible" with the information specified in the ontology. Given a query, we are interested in those answers to this query that depend only on the information in the ontology, i.e., that are obtained by evaluating the query over a database compatible with the ontology, but independently of which is the actually chosen database. In other words, we are interested in those answers to the query that are obtained for *all* possible databases (including infinite ones) that are models of the ontology. This corresponds to the fact that the ontology conveys only incomplete information about the domain of interest, and we want to guarantee that the answers to a query that we obtain are *certain*, independently of how we complete this incomplete information. This leads us to the following definition of *certain answers* to a query over an ontology.

Definition 2.8. *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A},id}$ ontology and q a UCQ over $\mathcal{O}$. A tuple $\mathbf{c}$ of constants appearing in $\mathcal{O}$ is a* certain answer *to q over $\mathcal{O}$, written $\mathbf{c} \in \mathit{cert}(q, \mathcal{O})$, if for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $\mathbf{c}^{\mathcal{I}} \in q^{\mathcal{I}}$.*

Answering a query q posed to an ontology $\mathcal{O}$ means exactly to compute the set of certain answers to q over $\mathcal{O}$.

Example 2.9. Consider again the ontology introduced in Example 2.3, and queries q_1 and q_2 introduced in Example 2.7. One can easily verify that

$$\begin{aligned} \mathit{cert}(q_1, \mathcal{O}) &= \{(\mathit{m8NT}^{\mathcal{I}}),\ (\mathit{m8RM}^{\mathcal{I}}),\ (\mathit{m7RJ}^{\mathcal{I}})\}, \\ \mathit{cert}(q_2, \mathcal{O}) &= \{(\mathit{roma}^{\mathcal{I}}, \texttt{"RJ"}),\ (\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\}. \end{aligned}$$

■

Notice that, in the case where $\mathcal{O}$ is an unsatisfiable ontology, the set of certain answers to a (U)CQ q is the finite set of all possible tuples of constants whose arity is the one of q. We denote such a set by $\mathit{AllTup}(q, \mathcal{O})$.

2.5 Reasoning Services

In studying *DL-Lite*$_{\mathcal{A},id}$, we are interested in several reasoning services, including the traditional DL reasoning services. Specifically, we consider the following problems for *DL-Lite*$_{\mathcal{A},id}$ ontologies:

- *Ontology satisfiability*, i.e., given an ontology $\mathcal{O}$, verify whether $\mathcal{O}$ admits at least one model.
- *Concept and role satisfiability*, i.e., given a TBox $\mathcal{T}$ and a concept C (resp., a role R), verify whether $\mathcal{T}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp., $R^{\mathcal{I}} \neq \emptyset$).
- We say that an ontology $\mathcal{O}$ (resp., a TBox $\mathcal{T}$) *logically implies* an assertion α, denoted $\mathcal{O} \models \alpha$ (resp., $\mathcal{T} \models \alpha$, if every model of $\mathcal{O}$ (resp., $\mathcal{T}$) satisfies α. The problem of *logical implication of assertions* consists of the following sub-problems:
  - *instance checking*, i.e., given an ontology $\mathcal{O}$, a concept C and a constant a (resp., a role R and a pair of constants a_1 and a_2), verify whether $\mathcal{O} \models C(a)$ (resp., $\mathcal{O} \models R(a_1, a_2)$);
  - *subsumption of concepts or roles*, i.e., given a TBox $\mathcal{T}$ and two general concepts C_1 and C_2 (resp., two general roles R_1 and R_2), verify whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{T} \models R_1 \sqsubseteq R_2$);
  - *checking functionality*, i.e., given a TBox $\mathcal{T}$ and a basic role Q, verify whether $\mathcal{T} \models (\mathsf{funct}\ Q)$.
  - *checking an identification constratins*, i.e., given a TBox $\mathcal{T}$ and an identification constraint $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$, verify whether $\mathcal{T} \models (\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$.

In addition we are interested in:

- *Query answering*, i.e., given an ontology $\mathcal{O}$ and a query q (either a CQ or a UCQ) over $\mathcal{O}$, compute the set $cert(q, \mathcal{O})$.

The following decision problem, called *recognition problem*, is associated to the query answering problem: given an ontology $\mathcal{O}$, a query q (either a CQ or a UCQ), and a tuple of constants $\boldsymbol{a}$ of $\mathcal{O}$, check whether $\boldsymbol{a} \in cert(q, \mathcal{O})$. When we talk about the computational complexity of query answering, in fact we implicitly refer to the associated recognition problem.

In analyzing the computational complexity of a reasoning problem over a DL ontology, we distinguish between data complexity and combined complexity [90]: *data complexity* is the complexity measured with respect to the size of the ABox only, while *combined complexity* is the complexity measured with respect to the size of all inputs to the problem, i.e., the TBox, the ABox, and the query. The data complexity measure is of interest in all those cases where the size of the intensional level of the ontology (i.e., the TBox) is negligible w.r.t. the size of the data (i.e., the ABox), as in ontology-based data access (cf. Section 1.3).

2.6 The Notion of FOL-Rewritability

We now introduce the notion of FOL-rewritability for both satisfiability and query answering, which will be used in the sequel.

First, given an ABox $\mathcal{A}$ (of the kind considered above), we denote by $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ *the interpretation* defined as follows:

- $\Delta^{DB(\mathcal{A})}$ is the non-empty set consisting of the union of the set of all object constants occurring in $\mathcal{A}$ and the set $\{val(c) \mid c$ is a value constant that occurs in $\mathcal{A}\}$,
- $a^{DB(\mathcal{A})} = a$, for each object constant a,
- $A^{DB(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept A,
- $P^{DB(\mathcal{A})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, for each atomic role P, and
- $U^{DB(\mathcal{A})} = \{(a, val(c)) \mid U(a, c) \in \mathcal{A}\}$, for each atomic attribute U.

Observe that the interpretation $DB(\mathcal{A})$ is a minimal model of the ABox $\mathcal{A}$.

Intuitively, FOL-rewritability of satisfiability (resp., query answering) captures the property that we can reduce satisfiability checking (resp., query answering) to evaluating a FOL query over the ABox $\mathcal{A}$ considered as a relational database, i.e., over $DB(\mathcal{A})$. The definitions follow.

Definition 2.10. *Satisfiability in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a boolean FOL query q, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$, the ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if q evaluates to false in $DB(\mathcal{A})$.*

Definition 2.11. *Answering UCQs in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every UCQ q and every TBox $\mathcal{T}$ expressed over $\mathcal{L}$, there exists a FOL query q_1, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$ and every tuple of constants $\boldsymbol{a}$ occurring in $\mathcal{A}$, we have that $\boldsymbol{a} \in cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ if and only if $\boldsymbol{a}^{DB(\mathcal{A})} \in q_1^{DB(\mathcal{A})}$.*

We remark that FOL-rewritability of a reasoning problem that involves the ABox of an ontology (such as satisfiability or query answering) is tightly related to the data complexity of the problem. Indeed, since the FOL query considered in the above definitions depends only on the TBox (and the query), but not on the ABox, and since the evaluation of a First-Order Logic query (i.e., an SQL query without aggregation) over an ABox is in AC^0 in data complexity [1], FOL-rewritability of a problem has as an immediate consequence that the problem is in AC^0 in data complexity. Hence, one way of showing that for a certain DL $\mathcal{L}$ a problem is *not* FOL-rewritable, is to show that the data complexity of the problem for the DL $\mathcal{L}$ is above AC^0, e.g., LOGSPACE-hard, NLOGSPACE-hard, PTIME-hard, or even coNP-hard. We will provide some results of this form in Section 6 (see also [4]).

3 UML Class Diagrams as an Ontology Language

In this section, we discuss how UML class diagrams can be considered as an ontology language, and we show how such diagrams can be captured in *DL-Lite*$_{\mathcal{A},id}$.

Since we concentrate on class diagrams from the conceptual perspective, we do not deal with those features that are more relevant for the software engineering perspective, such as operations (methods) associated to classes, or public, protected, and private qualifiers for methods and attributes. Also, for sake of brevity and to smooth the presentation we make some simplifying assumptions that could all be lifted without changing the results presented here (we refer to [11] for further details). In particular, we will not deal explicitly with associations of arity greater than 2, and we will only deal with the following multiplicities:

- unconstrained, i.e., $0..*$,
- functional participation, i.e., $0..1$,
- mandatory participation, i.e., $1..*$, and
- one-to-one correspondence, i.e., $1..1$.

These multiplicities are particularly important since they convey meaningful semantic aspects in modeling, and thus are the most commonly used ones.

Our goal is twofold. On the one hand, we aim at showing how class diagrams can be expressed in DLs. On the other hand, we aim at understanding which is the complexity of inference over an UML class diagram. We will show that the formalization in DLs helps us in deriving complexity results both for reasoning and for query answering over an UML class diagram.

3.1 Classes and Attributes

A *class* in a UML class diagram denotes a *sets of objects* with common features. The specification of a class contains its *name* and its *attributes*, each denoted by a name (possibly followed by the *multiplicity*, between square brackets) and with an associated *type*, which indicates the domain of the attribute values. A UML class is represented by a DL concept. This follows naturally from the fact that both UML classes and DL concepts denote *sets of objects*.

A UML *attribute* a of type T for a class C associates to each instance of C, zero, one, or more instances of type T. An optional *multiplicity* $[i..j]$ for a specifies that a associates to each instance of C, at least i and most j instances of T. When the multiplicity for an attribute is missing, $[1..1]$ is assumed, i.e., the attribute is *mandatory* and *single-valued*.

To formalize attributes, we have to think of an attribute a of type T for a class C as a binary relation between instances of C and instances of T. We capture such a binary relation by means of a DL attribute a_C. To specify the type of the attribute we use the DL assertions

$$\delta(a_C) \sqsubseteq C, \qquad\qquad \rho(a_C) \sqsubseteq T.$$

Such assertions specify precisely that, for each instance (c, v) of the attribute a_C, the object c is an instance of C, and the value v is an instance of T. Note that the attribute name a is not necessarily unique in the whole diagram, and hence two different classes, say C and C' could both have attribute a, possibly of different types. This situation is correctly captured in the DL formalization, where the attribute is contextualized to each class with a distinguished DL attribute, i.e., a_C and $a_{C'}$.

To specify that the attribute is mandatory (i.e., multiplicity $[1..*]$), we add the assertion

$$C \sqsubseteq \delta(a_C),$$

which specifies that each instance of C participates necessarily at least once to the DL attribute a_C. To specify that the attribute is single-valued (i.e., multiplicity $[0..1]$), we add the functionality assertion

$$(\mathsf{funct}\ a_C).$$

Finally, if the attribute is both mandatory and single-valued (i.e., multiplicity $[1..1]$), we use both assertions together, i.e.,

$$C \sqsubseteq \delta(a_C), \qquad (\mathsf{funct}\ a_C).$$

3.2 Associations

An *association* in UML is a relation between the instances of two (or more) classes. An association often has a related *association class* that describes properties of the association, such as attributes, operations, etc. A binary association A between the instances of two classes C_1 and C_2 is graphically rendered as in Figure 6(a), where the *multiplicity* $m_\ell..m_u$ specifies that each instance of class C_1 can participate at least m_ℓ times and at most m_u times to association A. The multiplicity $n_\ell..n_u$ has an analogous meaning for class C_2.

An association A between classes C_1 and C_2 is formalized in DL by means of a role A on which we enforce the assertions

$$\exists A \sqsubseteq C_1, \qquad \exists A^- \sqsubseteq C_2.$$

To express the multiplicity $m_\ell..m_u$ on the participation of instances of C_2 for each given instance of C_1, we use the assertion $C_1 \sqsubseteq \exists A$, if $m_\ell = 1$, and $(\mathsf{funct}\ A)$, if $m_u = 1$. We can use similar assertions for the multiplicity $n_\ell..n_u$ on the participation of instances of C_1 for each given instance of C_2, i.e., $C_2 \sqsubseteq \exists A^-$, if $n_\ell = 1$, and $(\mathsf{funct}\ A^-)$, if $n_u = 1$.

Next we focus on *associations* with a related *association class*, as shown in Figure 6(b), where the class A is the association class related to the association, and $R_{A,1}$ and $R_{A,2}$, if present, are the *role names* of C_1 and C_2 respectively, i.e., they specify the role that each class plays within the association A.

We formalize in DL an association A with an association class, by reifying it into a DL concept A and introducing two DL roles $R_{A,1}$, $R_{A,2}$, one for each role of A, which intuitively connect an object representing an instance of the association respectively to the instances of C_1 and C_2 that participate to the association[9]. Then, we enforce that each instance of A participates exactly once both to $R_{A,1}$ and to $R_{A,2}$, by means of the assertions

$$A \sqsubseteq \exists R_{A,1}, \qquad (\mathsf{funct}\ R_{A,1}), \qquad A \sqsubseteq \exists R_{A,2}, \qquad (\mathsf{funct}\ R_{A,2}).$$

To represent that the association A is between classes C_1 and C_2, we use the assertions

$$\exists R_{A,1} \sqsubseteq A, \qquad \exists R_{A,1}^- \sqsubseteq C_1, \qquad \exists R_{A,2} \sqsubseteq A, \qquad \exists R_{A,2}^- \sqsubseteq C_2.$$

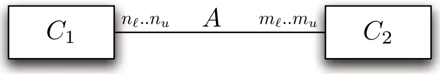
Finally, we use the assertion

$$(\mathsf{id}\ A\ R_{A,1}, R_{A,2})$$

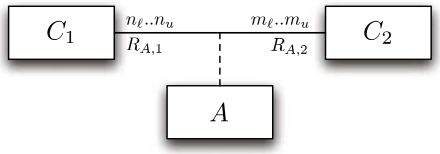
to specify that each instance of the concept A represents a *distinct* tuple in $C_1 \times C_2$.[10]

[9] If the roles of the association are not available, we may use an arbitrary DL role name.

[10] Notice that such an approach can immediately be used to represent an association of any arity: it suffices to repeat the above for every component.



(a) Without association class.



(b) With association class.

Fig. 6. Associations in UML

We can easily represent in DL multiplicities on an association with association class, by imposing suitable assertions on the inverses of the DL roles modeling the roles of the association. For example, to say that there is a one-to-one participation of instances of C_1 in the association (with related association class) A, we assert

$$C_1 \sqsubseteq \exists R_{A,1}^-, \qquad\qquad (\text{funct } R_{A,1}^-).$$

3.3 Generalizations and Class Hierarchies

In UML, one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class.

Generalization is naturally supported in DLs. If a UML class C_2 generalizes a class C_1, we can express this by the DL assertion

$$C_1 \sqsubseteq C_2.$$

Inheritance between DL concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of $\sqsubseteq$, which is based on the subset relation. As a consequence, in the formalization, each attribute of C_2 and each association involving C_2 is correctly inherited by C_1. Observe that the formalization in DL also captures directly multiple inheritance between classes.

In UML, one can group several generalizations into a *class hierarchy*, as shown in Figure 7. Such a hierarchy is captured in DL by a set of inclusion assertions, one between each child class and the parent class, i.e.,

$$C_i \sqsubseteq C, \qquad \text{for each } i \in \{1, \ldots, n\}.$$

Often, when defining generalizations between classes, we need to add additional assertions among the involved classes. For example, for the class hierarchy in Figure 7, an assertion may express that $C_1, \ldots, C_n$ are *mutually disjoint*. In DL, such a relationship can be expressed by the assertions

$$C_i \sqsubseteq \neg C_j, \qquad \text{for each } i, j \in \{1, \ldots, n\} \text{ with } i \neq j.$$

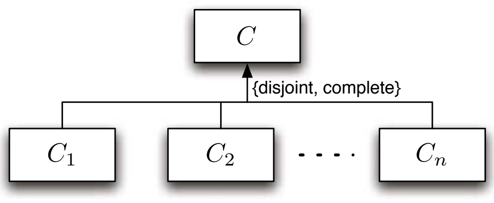


Fig. 7. A class hierarchy in UML

Moreover, we may want to express that a generalization hierarchy is *complete*, i.e., that the subclasses $C_1, \ldots, C_n$ are a *covering* of the superclass C. We can represent such a situation in DL by including the additional assertion

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_n.$$

Such an assertion models a form of *disjunctive information*: each instance of C is either an instance of C_1, or an instance of C_2, ... or an instance of C_n. Notice, however, that the use of concept disjunction, and hence of the inclusion assertion above, is not allowed in *DL-Lite*$_{\mathcal{A},id}$.

3.4 Subset Assertions between Associations

Similarly to generalization between classes, UML allows one to state *subset assertions* between associations. A subset assertion between two associations A_1 and A_2 can be modeled in DL by means of the role inclusion assertion

$$A_1 \sqsubseteq A_2,$$

involving the two DL roles A_1 and A_2 representing the associations. Notice that this is allowed in *DL-Lite*$_{\mathcal{A},id}$ only if none of the maximum multiplicities of the two classes participating to A_2 is equal to 1.

With respect to a generalization between two association classes A_1 and A_2, we note that to correctly capture the corresponding subset assertion between the associations represented by the association classes, we would need to introduce not only inclusion assertions between the concepts representing the association classes, but also between the DL roles representing corresponding roles of A_1 and A_2. Consider, for example, the generalization between association classes depicted in Figure 8. We can correctly capture the two associations with association classes A_1 and A_2 by the following DL assertions:

$$\begin{array}{llll}
A_1 \sqsubseteq \exists R_{A1,1}, & (\mathsf{funct}\ R_{A1,1}), & A_2 \sqsubseteq \exists R_{A2,1}, & (\mathsf{funct}\ R_{A2,1}), \\
A_1 \sqsubseteq \exists R_{A1,2}, & (\mathsf{funct}\ R_{A1,2}), & A_2 \sqsubseteq \exists R_{A2,2}, & (\mathsf{funct}\ R_{A2,2}), \\
\exists R_{A1,1} \sqsubseteq A_1, & \exists R^-_{A1,1} \sqsubseteq C_{11}, & \exists R_{A2,1} \sqsubseteq A_2, & \exists R^-_{A2,1} \sqsubseteq C_{21}, \\
\exists R_{A1,2} \sqsubseteq A_1, & \exists R^-_{A1,2} \sqsubseteq C_{12}, & \exists R_{A2,2} \sqsubseteq A_2, & \exists R^-_{A2,2} \sqsubseteq C_{22}, \\
\multicolumn{2}{c}{(\mathsf{id}\ A_1\ R_{A1,1}, R_{A1,2}),} & \multicolumn{2}{c}{(\mathsf{id}\ A_2\ R_{A2,1}, R_{A2,2}).}
\end{array}$$

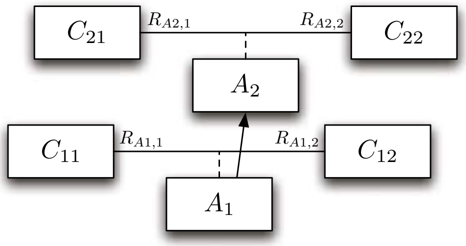


Fig. 8. A generalization between association classes in UML

Finally, to capture the generalization, we could use the following inclusion assertions

$$A_1 \sqsubseteq A_2, \qquad R_{A1,1} \sqsubseteq R_{A2,1}, \qquad R_{A1,2} \sqsubseteq R_{A2,2}.$$

However, since $R_{A2,1}$ and $R_{A2,2}$ are functional roles (and are also used in an identification assertion), we actually cannot specialize them, if we want to stay in *DL-Lite*$_{\mathcal{A},id}$. Hence, generalization of associations with association classes in general cannot be formalized in *DL-Lite*$_{\mathcal{A},id}$.

Finally, we observe that a formalization in *DL-Lite*$_{\mathcal{A},id}$ of generalization between association classes is possible if the sub-association does not specify new classes for the domain and range of the association with respect to the super-association. In the example of Figure 8 this would mean that C_{11} coincides with C_{21} and C_{12} coincides with C_{22}. In this case, the sub-association A_1 is represented by simply using the same DL roles as for A_2 to denote its components. Hence, it is not necessary to introduce an inclusion assertion between functional DL roles to correctly capture the generalization between association classes.

3.5 Reasoning and Query Answering over UML Class Diagrams

The fact that UML class diagrams can be captured by DLs enables the possibility of performing sound and complete reasoning to do formal verification at design time and query answering at runtime, as will be illustrated in the next sections. Hence, one can exploit such ability to get support during the design phase of an ontology-based data access system, and to take the information in the UML class diagram fully into account during query answering.

It was shown in [11] that, unfortunately, reasoning (in particular checking the consistency of the diagram, a task to which other typical reasoning tasks of interest reduce) is EXPTIME-hard. What this result tells us is that, if the TBox is expressed in UML, then the support at design time for an ontology-based data access system may be difficult to obtain if the schema has a reasonable size.

Turning to query answering, the situation is even worse. The results in Section 6 imply that answering conjunctive queries in the presence of a UML class diagram formed by a single generalization with covering assertion is coNP-hard in the size of the instances of classes and associations. Hence, query answering over even moderately large data sets is again infeasible in practice. It is not difficult to see that this implies that, in an ontology-based data access system where the TBox is expressed as a UML diagram, answering conjunctive queries is coNP-hard with respect to the size of the data in the accessed source.

Actually, as we will see in Section 6, one culprit of such a high complexity is the ability of expressing covering assertions, which induces reasoning by cases. A further cause of high complexity is the unrestricted interaction between multiplicities (actually, functionality) and subset constraints between associations [22,4]. Once we disallow covering and suitably restrict the simultaneous use of subset constraints between associations and multiplicities, not only the sources of exponential complexity disappear, but actually query answering becomes reducible to standard SQL evaluation over a relational database, as will be demonstrated in the following.

4 Reasoning over Ontologies

In this section, we study traditional DL reasoning services for ontologies. In particular, we consider the reasoning services described in Section 2.5, and we show that all such reasoning services are in PTime w.r.t. combined complexity, and that instance checking and satisfiability (which make use of the ABox) are FOL-rewritable, and hence in AC^0 with respect to data complexity. We concentrate in this section on *DL-Lite*$_{\mathcal{A}}$ ontologies and address the addition of identification assertions in Section 5.6, after having discussed in Section 5 query answering based on reformulation. We deal first with ontology satisfiability, and then we tackle concept and role satisfiability, and logical implication. In fact, we will show that the latter reasoning services can be basically reduced to ontology satisfiability. Finally, we provide the complexity results mentioned above.

In the following, to ease the presentation, we make several simplifying assumptions that however do not affect the generality of the presented results:

1. Since the distinction between objects and values does not have an impact on the ontology reasoning services, we will deal only with ontologies that contain object constants, concepts, and roles only, and do not consider value constants, value domains, and attributes. Hence, we also rule out concepts of the form $\delta(U)$ and $\delta_F(U)$. With respect to the semantics, since we don't have to deal with values, we consider only interpretations $\mathcal{I}$ where the domain of values $\Delta_V^{\mathcal{I}}$ is empty, hence the interpretation domain $\Delta^{\mathcal{I}}$ coincides with the domain of objects $\Delta_O^{\mathcal{I}}$.
2. We will assume that the ontology does not contain qualified existential restrictions, i.e., concepts of the form $\exists Q.C$. Notice that in *DL-Lite*$_{\mathcal{A},id}$ such concepts may appear only in the right-hand side of inclusion assertions of the form $B \sqsubseteq \exists Q.C$, and only if the role Q and its inverse do not appear in a functionality assertion. We can replace the inclusion assertion $B \sqsubseteq \exists Q.C$ by the inclusion assertions

$$\begin{aligned} B &\sqsubseteq \exists P_{new} \\ \exists P_{new}^- &\sqsubseteq C \\ P_{new} &\sqsubseteq Q \end{aligned}$$

where P_{new} is a fresh atomic role. It is easy to see that the resulting ontology preserves all reasoning services over the original ontology. By repeated application of the above transformation, once for each occurrence of a concept $\exists Q.C$ in the ontology obtained from the previous application[11], we obtain an ontology that does not contain qualified existential restrictions and that preserves all reasoning services over the original ontology.

3. Since inclusion assertions of the form $B \sqsubseteq \top_c$ do not have an impact on the semantics, we can simply discard them in reasoning.

Hence, in the following, we will consider the following simplified grammar for *DL-Lite*$_\mathcal{A}$ expressions:

$$\begin{array}{rclcrcl} B & \longrightarrow & A \mid \exists Q & \qquad & Q & \longrightarrow & P \mid P^- \\ C & \longrightarrow & B \mid \neg B & & R & \longrightarrow & Q \mid \neg Q \end{array}$$

Our first goal is to show that ontology satisfiability is FOL-rewritable. To this aim, we resort to two main constructions, namely the canonical interpretation and the closure of the negative inclusions, which we present below.

We recall that assertions of the form $B_1 \sqsubseteq B_2$ or $Q_1 \sqsubseteq Q_2$ are called *positive inclusions (PIs)*, and assertions of the form $B_1 \sqsubseteq \neg B_2$ or $Q_1 \sqsubseteq \neg Q_2$ are called *negative inclusions (NIs)*. Notice that due to the simplified form of the grammar that we are adopting here, these are the only kinds of inclusion assertions that we need to consider.

4.1 Canonical Interpretation

The canonical interpretation of a *DL-Lite*$_\mathcal{A}$ ontology is an interpretation constructed according to the notion of *chase* [1]. In particular, we adapt here the notion of *restricted chase* adopted by Johnson and Klug in [56].

We start by defining the notion of applicable positive inclusion assertions (PIs), and then we exploit applicable PIs to construct the chase for a *DL-Lite*$_\mathcal{A}$ ontology. Finally, with the notion of chase in place, we give the definition of canonical interpretation.

In the following, for easiness of exposition, we make use of the following notation for a basic role Q and two constants a_1 and a_2:

$$Q(a_1, a_2) \text{ denotes } \begin{cases} P(a_1, a_2), & \text{if } Q = P, \\ P(a_2, a_1), & \text{if } Q = P^-. \end{cases}$$

Definition 4.1. *Let $\mathcal{S}$ be a set of DL-Lite$_\mathcal{A}$ membership assertions. Then, a PI α is* applicable *in $\mathcal{S}$ to a membership assertion $\beta \in \mathcal{S}$ if*

- $\alpha = A_1 \sqsubseteq A_2$, $\beta = A_1(a)$, *and* $A_2(a) \notin \mathcal{S}$;
- $\alpha = A \sqsubseteq \exists Q$, $\beta = A(a)$, *and there does not exist any constant* a' *such that* $Q(a, a') \in \mathcal{S}$;
- $\alpha = \exists Q \sqsubseteq A$, $\beta = Q(a, a')$, *and* $A(a) \notin \mathcal{S}$;
- $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$, $\beta = Q_1(a_1, a_2)$, *and there does not exist any constant* a_2' *such that* $Q_2(a_1, a_2') \in \mathcal{S}$;
- $\alpha = Q_1 \sqsubseteq Q_2$, $\beta = Q_1(a_1, a_2)$, *and* $Q_2(a_1, a_2) \notin \mathcal{S}$.

[11] Note that an ontology may contain concepts in which qualified existential restrictions are nested within each other.

Applicable PIs can be used, i.e., *applied*, in order to construct the chase of an ontology. Roughly speaking, the chase of a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a (possibly infinite) set of membership assertions, constructed step-by-step starting from the ABox $\mathcal{A}$. At each step of the construction, a PI $\alpha \in \mathcal{T}$ is applied to a membership assertion β belonging to the current set $\mathcal{S}$ of membership assertions. Applying a PI means adding a new suitable membership assertion to $\mathcal{S}$, thus obtaining a new set $\mathcal{S}'$ in which α is not applicable to β anymore. For example, if $\alpha = A_1 \sqsubseteq A_2$ is applicable in $\mathcal{S}$ to $\beta = A_1(a)$, the membership assertion to be added to $\mathcal{S}$ is $A_2(a)$, i.e., $\mathcal{S}' = \mathcal{S} \cup A_2(a)$. In some cases (i.e., $\alpha = A \sqsubseteq \exists Q$ or $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$), to achieve an analogous aim, the new membership assertion has to make use of a new constant symbol that does not occur in $\mathcal{S}$.

Notice that such a construction process strongly depends on the order in which we select both the PI to be applied at each step and the membership assertion to which such a PI is applied, as well as on which constants we introduce at each step. Therefore, a number of syntactically distinct sets of membership assertions might result from this process. However, it is possible to show that the result is unique up to renaming of constants occurring in each such a set. Since we want our construction process to result in a unique chase of a certain ontology, along the lines of [56], we assume in the following to have a fixed infinite set of constants, whose symbols are ordered in lexicographic way, and we select PIs, membership assertions and constant symbols in lexicographic order. More precisely, given a ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote with Γ_A the set of all constant symbols occurring in $\mathcal{A}$. Also, we assume to have an infinite set Γ_N of constant symbols not occurring in $\mathcal{A}$, such that the set $\Gamma_C = \Gamma_A \cup \Gamma_N$ is totally ordered in lexicographic way. Then, our notion of chase is precisely given below.

Definition 4.2. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A}}$ ontology, let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$, let n be the number of membership assertions in $\mathcal{A}$, and let Γ_N be the set of constants defined above. Assume that the membership assertions in $\mathcal{A}$ are numbered from 1 to n following their lexicographic order, and consider the following definition of sets S_j of membership assertions:*

- $\mathcal{S}_0 = \mathcal{A}$
- $\mathcal{S}_{j+1} = \mathcal{S}_j \cup \{\beta_{new}\}$, *where β_{new} is a membership assertion numbered with $n + j + 1$ in $\mathcal{S}_{j+1}$ and obtained as follows:*

```
let $\beta$ be the first membership assertion in $\mathcal{S}_j$ such that there exists a PI $\alpha \in \mathcal{T}_p$
    applicable in $\mathcal{S}_j$ to $\beta$
let $\alpha$ be the lexicographically first PI applicable in $\mathcal{S}_j$ to $\beta$
let $a_{new}$ be the constant of $\Gamma_N$ that follows lexicographically all constants in $\mathcal{S}_j$
case $\alpha$, $\beta$ of
    (cr1) $\alpha = A_1 \sqsubseteq A_2$            and $\beta = A_1(a)$      then $\beta_{new} = A_2(a)$
    (cr2) $\alpha = A \sqsubseteq \exists Q$         and $\beta = A(a)$        then $\beta_{new} = Q(a, a_{new})$
    (cr3) $\alpha = \exists Q \sqsubseteq A$         and $\beta = Q(a, a')$    then $\beta_{new} = A(a)$
    (cr4) $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$ and $\beta = Q_1(a, a')$ then $\beta_{new} = Q_2(a, a_{new})$
    (cr5) $\alpha = Q_1 \sqsubseteq Q_2$            and $\beta = Q_1(a, a')$  then $\beta_{new} = Q_2(a, a')$.
```

Then, we call chase of $\mathcal{O}$*, denoted* $chase(\mathcal{O})$*, the set of membership assertions obtained as the infinite union of all* $\mathcal{S}_j$*, i.e.,*

$$chase(\mathcal{O}) = \bigcup_{j \in \mathbb{N}} \mathcal{S}_j.$$

In the above definition, **cr1**, **cr2**, **cr3**, **cr4**, and **cr5** indicate the five rules that are used for constructing the chase, each one corresponding to the application of a PI. Such rules are called *chase rules*, and we say that a chase rule is *applied to* a membership assertion β if the corresponding PI is applied to β. Observe also that NIs and functionality assertions in $\mathcal{O}$ have no role in constructing $chase(\mathcal{O})$. Indeed $chase(\mathcal{O})$ depends only on the ABox $\mathcal{A}$ and the PIs in $\mathcal{T}$.

In the following, we will denote with $chase_i(\mathcal{O})$ the portion of the chase obtained after i applications of the chase rules, selected according to the ordering established in Definition 4.2, i.e.,

$$chase_i(\mathcal{O}) = \bigcup_{j \in \{0,..,i\}} \mathcal{S}_j.$$

The following property shows that the notion of chase of an ontology is fair.

Proposition 4.3. *Let* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *be a DL-Lite*$_{\mathcal{A}}$ *ontology, and let* α *be a PI in* $\mathcal{T}$*. Then, if there is an* $i \in \mathbb{N}$ *such that* α *is applicable in* $chase_i(\mathcal{O})$ *to a membership assertion* $\beta \in chase_i(\mathcal{O})$*, then there is a* $j \geq i$ *such that* $chase_{j+1}(\mathcal{O}) = chase_j(\mathcal{O}) \cup \beta'$*, where* β' *is the result of applying* α *to* β *in* $chase_j(\mathcal{O})$*.*

Proof. Assume by contradiction that there is no $j \geq i$ such that $chase_{j+1}(\mathcal{O}) = chase_j(\mathcal{O}) \cup \beta'$. This would mean that either there are infinitely many membership assertions that precede β in the ordering that we choose for membership assertions in $chase(\mathcal{O})$, or that there are infinitely many chase rules applied to some membership assertion that precedes β. However, none of these cases is possible. Indeed, β is assigned with an ordering number m such that exactly $m - 1$ membership assertions precede β. Furthermore, a PI can be applied at most once to a membership assertion (afterwards, the precondition is not satisfied and the PI is not applicable anymore), and also there exists only a finite number ℓ of PIs. Therefore, it is possible to apply a chase rule to some membership assertion at most ℓ times. We can thus conclude that the claim holds. $\square$

With the notion of chase in place we can introduce the notion of canonical interpretation.

Definition 4.4. *The* canonical interpretation $can(\mathcal{O}) = \langle \Delta^{can(\mathcal{O})}, \cdot^{can(\mathcal{O})} \rangle$ *is the interpretation where:*

- $\Delta^{can(\mathcal{O})} = \Gamma_C$,
- $a^{can(\mathcal{O})} = a$*, for each constant* a *occurring in* $chase(\mathcal{O})$*,*
- $A^{can(\mathcal{O})} = \{a \mid A(a) \in chase(\mathcal{O})\}$*, for each atomic concept* A*, and*
- $P^{can(\mathcal{O})} = \{(a_1, a_2) \mid P(a_1, a_2) \in chase(\mathcal{O})\}$*, for each atomic role* P*.*

We also define $can_i(\mathcal{O}) = \langle \Delta^{can(\mathcal{O})}, \cdot^{can_i(\mathcal{O})} \rangle$*, where* $\cdot^{can_i(\mathcal{O})}$ *is analogous to* $\cdot^{can(\mathcal{O})}$*, except that it refers to* $chase_i(\mathcal{O})$ *instead of* $chase(\mathcal{O})$*.*

According to the above definition, it is easy to see that $can(\mathcal{O})$ (resp., $can_i(\mathcal{O})$) is unique. Notice also that $can_0(\mathcal{O})$ is tightly related to the interpretation $DB(\mathcal{A})$. Indeed, while $\Delta^{DB(\mathcal{A})} \subseteq \Delta^{can(\mathcal{O})}$, we have that $\cdot^{DB(\mathcal{A})} = \cdot^{can_0(\mathcal{O})}$.

We point out that $chase(\mathcal{O})$ and $can(\mathcal{O})$ (resp., $chase_i(\mathcal{O})$) and $can_i(\mathcal{O})$) are strongly connected. In particular, we note that, whereas $chase_{i+1}(\mathcal{O})$ is obtained by adding a membership assertion to $chase_i(\mathcal{O})$, $can_{i+1}(\mathcal{O})$ can be seen as obtained from $can_i(\mathcal{O})$ by adding either an object to the extension of an atomic concept of $\mathcal{O}$, or a pair of objects to the extension of an atomic role of $\mathcal{O}$ (notice that the domain of interpretation is the same in each $can_i(\mathcal{O})$, and in particular in $can(\mathcal{O})$). By virtue of the strong connection discussed above, in the following we will often prove properties of $can(\mathcal{O})$ (resp., $can_i(\mathcal{O})$) by reasoning over the structure of $chase(\mathcal{O})$ (resp., $chase_i(\mathcal{O})$).

Now, we are ready to show a notable property that holds for $can(\mathcal{O})$.

Lemma 4.5. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$.*

Proof. Since $\langle \mathcal{T}_p, \mathcal{A} \rangle$ does not contain NIs and functionality assertions, to prove the claim we only need to show that $can(\mathcal{O})$ satisfies all membership assertions in $\mathcal{A}$ and all PIs in $\mathcal{T}_p$. The fact that $can(\mathcal{O})$ satisfies all membership assertions in $\mathcal{A}$ follows from the fact that $\mathcal{A} \subseteq chase(\mathcal{O})$. Then, it remains to prove that $can(\mathcal{O}) \models \mathcal{T}_p$. Let us proceed by contradiction, considering all possible cases:

1. Assume by contradiction that a PI of the form $A_1 \sqsubseteq A_2 \in \mathcal{T}_p$, where A_1 and A_2 are atomic concepts, is not satisfied by $can(\mathcal{O})$. This means that there exists a constant $a \in \Gamma_C$ such that $A_1(a) \in chase(\mathcal{O})$ and $A_2(a) \notin chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr1**, since $A_1 \sqsubseteq A_2$ would be applicable to $A_1(a)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $A_2(a)$ in $chase(\mathcal{O})$. This contradicts the assumption.
2. Assume by contradiction that a PI of the form $A \sqsubseteq \exists Q \in \mathcal{T}_p$, where A is an atomic concept and Q is a basic role, is not satisfied by $can(\mathcal{O})$. This means that there exists a constant $a \in \Gamma_C$ such that $A(a) \in chase(\mathcal{O})$ and there does not exist a constant $a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr2**, since $A \sqsubseteq \exists Q$ would be applicable to $A(a)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q(a, a_2)$ in $chase(\mathcal{O})$, where $a_2 \in \Gamma_C$ follows lexicographically all constants occurring in $chase(\mathcal{O})$ before the execution of **cr2**. This contradicts the assumption.
3. Assume by contradiction that a PI of the form $\exists Q \sqsubseteq A \in \mathcal{T}_p$, where Q is a basic role and A is an atomic concept, is not satisfied by $can(\mathcal{O})$. This means that there exists a pair of constants $a, a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase(\mathcal{O})$ and $A(a) \notin chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr3**, since $\exists Q \sqsubseteq A$ would be applicable to $Q(a, a_1)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $A(a)$ in $chase(\mathcal{O})$. This contradicts the assumption.
4. Assume by contradiction that a PI of the form $\exists Q_1 \sqsubseteq \exists Q_2 \in \mathcal{T}_p$, where Q_1 and Q_2 are basic roles, is not satisfied by $can(\mathcal{O})$. This means that there exists a pair

of constants $a, a_1 \in \Gamma_C$ such that $Q_1(a, a_1) \in \mathit{chase}(\mathcal{O})$ and there does not exist a constant $a_2 \in \Gamma_C$ such that $Q_2(a, a_2) \in \mathit{chase}(\mathcal{O})$. However, such a situation would trigger the chase rule **cr4** since $\exists Q_1 \sqsubseteq \exists Q_2$ would be applicable to $Q_1(a, a_1)$ in $\mathit{chase}(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q(a, a_3)$ in $\mathit{chase}(\mathcal{O})$, where $a_3 \in \Gamma_C$ follows lexicographically all constants occurring in $\mathit{chase}(\mathcal{O})$ before the execution of **cr4**. This contradicts the assumption.

5. Assume by contradiction that a PI of the form $Q_1 \sqsubseteq Q_2 \in \mathcal{T}_p$, where Q_1 and Q_2 are basic roles, is not satisfied by $\mathit{can}(\mathcal{O})$. This means that there exists a pair of constants $a, a_1 \in \Gamma_C$ such that $Q_1(a, a_1) \in \mathit{chase}(\mathcal{O})$ and $Q_2(a, a_1) \notin \mathit{chase}(\mathcal{O})$. However, such a situation would trigger the chase rule **cr5**, since $Q_1 \sqsubseteq Q_2$ would be applicable to $Q_1(a, a_1)$ in $\mathit{chase}(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q_2(a, a_1)$ in $\mathit{chase}(\mathcal{O})$. This contradicts the assumption. □

As a consequence of Lemma 4.5, every *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ with only positive inclusions in the TBox, i.e., such that $\mathcal{T} = \mathcal{T}_p$, is always satisfiable, since we can always construct $\mathit{can}(\mathcal{O})$, which is a model for $\mathcal{O}$. Now, one might ask if and how $\mathit{can}(\mathcal{O})$ can be exploited for checking the satisfiability of an ontology with also negative inclusions and functionality assertions.

As for functionality assertions, the following lemma shows that, to establish that they are satisfied by $\mathit{can}(\mathcal{O})$, we have to simply verify that the interpretation $\mathit{DB}(\mathcal{A})$ satisfies them (and vice-versa).

Lemma 4.6. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology, and let $\mathcal{T}_f$ be the set of functionality assertions in $\mathcal{T}$. Then, $\mathit{can}(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ if and only if $\mathit{DB}(\mathcal{A})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$.*

Proof. "$\Rightarrow$" We show that $\mathit{DB}(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$ if $\mathit{can}(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$. This can be easily seen by observing that $\mathcal{A} \subseteq \mathit{chase}(\mathcal{O})$, and therefore if a membership assertion in $\mathcal{A}$ or a functionality assertion in $\mathcal{T}_f$ is satisfied by $\mathit{can}(\mathcal{O})$, it is also satisfied by $\mathit{DB}(\mathcal{A})$ (notice in particular that $\Delta^{\mathit{DB}(\mathcal{A})} \subseteq \Delta^{\mathit{can}(\mathcal{O})}$).

"$\Leftarrow$" We show that $\mathit{can}(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$ if $\mathit{DB}(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$. By virtue of the correspondence between $\mathit{can}(\mathcal{O})$ and $\mathit{chase}(\mathcal{O})$, we proceed by induction on the construction of $\mathit{chase}(\mathcal{O})$.

Base step. We have that $\mathit{chase}_0(\mathcal{O}) = \mathcal{A}$, and since $\mathit{DB}(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$, it follows that $\mathit{can}_0(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$.

Inductive step. Let us assume by contradiction that for some $i \geq 0$, $\mathit{can}_i(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ and $\mathit{can}_{i+1}(\mathcal{O})$ is not. Notice that **cr2**, **cr4**, and **cr5** are the only rules that introduce new role instances, and thus may lead to a violation of a functionality assertion in $\mathit{can}_{i+1}(\mathcal{O})$. However, due to the restriction on the interaction between functionality and role inclusion assertions in *DL-Lite*$_\mathcal{A}$, rule **cr5** will never be applied if $\mathcal{T}$ contains a functionality assertion for Q_2 or its inverse. Thus, we need to consider only the rules **cr2** and **cr4**. Let us consider first rule **cr2**, and assume that $\mathit{chase}_{i+1}(\mathcal{O})$ is obtained by applying **cr2** to $\mathit{chase}_i(\mathcal{O})$. This means that a PI of the form $A \sqsubseteq \exists Q$, where A is an atomic concept and Q is a basic role, is applied in $\mathit{chase}_i(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exists $a_1 \in \Gamma_C$ such that

$Q(a, a_1) \in chase_i(\mathcal{O})$. Therefore, $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup Q(a, a_{new})$, where $a_{new} \in \Gamma_C$ follows lexicographically all constants occurring in $chase_i(\mathcal{O})$. Now, if $can_{i+1}(\mathcal{O})$ is not a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$, there must exist (at least) a functionality assertion α that is not satisfied by $can_{i+1}(\mathcal{O})$.

- In the case where $\alpha = (\mathsf{funct}\ Q)$, for α to be violated, there must exist two pairs of objects (x, y) and (x, z) in $Q^{can_{i+1}(\mathcal{O})}$ such that $y \neq z$. However, we have that $(a, a_{new}) \in Q^{can_{i+1}(\mathcal{O})}$ and $a \notin \exists Q^{can_i(\mathcal{O})}$, since by applicability of $A \sqsubseteq \exists Q$ in $chase_i(\mathcal{O})$ it follows that there does not exist a constant $a' \in \Gamma_C$ such that $Q(a, a') \in chase_i(\mathcal{O})$. Therefore, there exists no pair $(a, a') \in Q^{can_{i+1}(\mathcal{O})}$ such that $a' \neq a_{new}$. Hence, we would conclude that (x, y) and (x, z) are in $Q^{can_i(\mathcal{O})}$, which would lead to a contradiction.
- In the case where $\alpha = (\mathsf{funct}\ Q^-)$, for α to be violated, there must exist two pairs of objects (y, x) and (z, x) in $Q^{can_{i+1}(\mathcal{O})}$ such that $y \neq z$. Since a_{new} is a fresh constant, not occurring in $chase_i(\mathcal{O})$, we can conclude that there exists no pair (a', a_{new}), with $a' \neq a$, such that $Q(a', a_{new}) \in chase_i(\mathcal{O})$, and therefore, there exists no pair $(a', a_{new}) \in Q^{can_{i+1}(\mathcal{O})}$. Hence, we would conclude that (y, x) and (z, x) are in $Q^{can_i(\mathcal{O})}$, which would lead to a contradiction.
- In the case in which $\alpha = (\mathsf{funct}\ Q')$, with $Q' \neq Q$ and $Q' \neq Q^-$, we would conclude that α is not satisfied already in $can_i(\mathcal{O})$, which would lead to a contradiction.

With an almost identical argument we can prove the inductive step also in the case in which $chase_{i+1}(\mathcal{O})$ is obtained by applying **cr4** to $chase_i(\mathcal{O})$. $\square$

4.2 Closure of Negative Inclusion Assertions

Let us now consider negative inclusions. In particular, we look for a property which is analogous to Lemma 4.6 for the case of NIs. Notice that, in this case, even if $DB(\mathcal{A})$ satisfies the NIs asserted in the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we have that $can(\mathcal{O})$ may not satisfy $\mathcal{O}$. For example, if $\mathcal{T}$ contains the inclusion assertions $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq \neg A_3$, and $\mathcal{A}$ contains the membership assertions $A_1(a)$ and $A_3(a)$, it is easy to see that $DB(\mathcal{A}) \models A_2 \sqsubseteq \neg A_3$, but $can(\mathcal{O}) \not\models A_2 \sqsubseteq \neg A_3$. However, as suggested by the simple example above, we get that to find the property we are looking for, we need to properly take into account the interaction between positive and negative inclusions. To this aim we construct a special TBox by closing the NIs with respect to the PIs.

Definition 4.7. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox. We call* NI-closure of $\mathcal{T}$*, denoted by $cln(\mathcal{T})$, the TBox defined inductively as follows:*

(1) all functionality assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
(2) all negative inclusion assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
(3) if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ is in $cln(\mathcal{T})$, then also $B_1 \sqsubseteq \neg B_3$ is in $cln(\mathcal{T})$;
(4) if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $\exists Q_2 \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists Q_2$ is in $cln(\mathcal{T})$, then also $\exists Q_1 \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;

(5) if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $\exists Q_2^- \sqsubseteq \neg B$ or $B \sqsubseteq \neg\exists Q_2^-$ is in $cln(\mathcal{T})$, then also $\exists Q_1^- \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;
(6) if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $Q_2 \sqsubseteq \neg Q_3$ or $Q_3 \sqsubseteq \neg Q_2$ is in $cln(\mathcal{T})$, then also $Q_1 \sqsubseteq \neg Q_3$ is in $cln(\mathcal{T})$.
(7) if one of the assertions $\exists Q \sqsubseteq \neg\exists Q$, $\exists Q^- \sqsubseteq \neg\exists Q^-$, or $Q \sqsubseteq \neg Q$ is in $cln(\mathcal{T})$, then all three such assertions are in $cln(\mathcal{T})$.

The following lemma shows that $cln(\mathcal{T})$ does not imply new negative inclusions or new functionality assertions not implied by $\mathcal{T}$.

Lemma 4.8. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and α a negative inclusion assertion or a functionality assertion. We have that, if $cln(\mathcal{T}) \models \alpha$, then $\mathcal{T} \models \alpha$.*

Proof. To prove the claim it is sufficient to observe that all assertions contained in $cln(\mathcal{T})$ are logically implied by $\mathcal{T}$. □

We are now ready to show that, provided we have computed $cln(\mathcal{T})$, the analogous of Lemma 4.6 holds also for NIs.

Lemma 4.9. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology. Then, $can(\mathcal{O})$ is a model of $\mathcal{O}$ if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

Proof. "⇒" By construction, $DB(\mathcal{A})$ cannot contradict a membership assertion in $\mathcal{A}$. Moreover, since $can(\mathcal{O})$ is a model of $\mathcal{O}$ and, by Lemma 4.8, each assertion in $cln(\mathcal{T})$ is logically implied by $\mathcal{O}$, we have that $can(\mathcal{O})$ is a model of $cln(\mathcal{T})$. Notice that $A^{DB(\mathcal{A})} = A^{can_0(\mathcal{O})} \subseteq A^{can(\mathcal{O})}$ for every atomic concept A in $\mathcal{O}$, and similarly $P^{DB(\mathcal{A})} = P^{can_0(\mathcal{O})} \subseteq P^{can(\mathcal{O})}$ for every atomic role P in $\mathcal{O}$. Now, considering that the structure of NIs and of functionality assertions is such that they cannot be contradicted by restricting the extension of atomic concepts and roles, we can conclude that $DB(\mathcal{A})$ is a model of $cln(\mathcal{T})$.

"⇐" We now prove that if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$, then $can(\mathcal{O})$ is a model of $\mathcal{O}$. From Lemma 4.5 it follows that $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$, where $\mathcal{T}_p$ is the set of PIs in $\mathcal{T}$. Moreover, since the set $\mathcal{T}_f$ of functionality assertions in $\mathcal{O}$ is contained in $cln(\mathcal{T})$, from Lemma 4.6 it follows that $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$. Hence, it remains to prove that $can(\mathcal{O})$ is a model of $\langle \mathcal{T} \setminus (\mathcal{T}_p \cup \mathcal{T}_f), \mathcal{A} \rangle$. We show this by proving that $can(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}) \setminus \mathcal{T}_f, \mathcal{A} \rangle$ (notice that $\mathcal{T} \setminus \mathcal{T}_p$ is contained in $cln(\mathcal{T})$). The proof is by induction on the construction of $chase(\mathcal{O})$.

Base step. By construction, $chase_0(\mathcal{O}) = \mathcal{A}$, and therefore $A^{can_0(\mathcal{O})} = A^{DB(\mathcal{A})}$ for every atomic concept A in $\mathcal{O}$, and $P^{can_0(\mathcal{O})} = P^{DB(\mathcal{A})}$ for every atomic role P in $\mathcal{O}$. Hence, by the assumption that $DB(\mathcal{A}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$, it follows that $can_0(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.

Inductive step. Let us assume by contradiction that $can_i(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$ and $can_{i+1}(\mathcal{O})$ is not, and that $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by execution of the rule **cr1**. According to **cr1**, a PI of the form $A_1 \sqsubseteq A_2$, where A_1 and A_2 are atomic concepts in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion of the form $A_1(a)$, such that $A_2(a) \notin chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{A_2(a)\}$ (notice that this means that $a \in A_2^{can_{i+1}(\mathcal{O})}$). Now, if $can_{i+1}(\mathcal{O})$

is not a model of $cln(\mathcal{T})$, there must exist a NI in $cln(\mathcal{T})$ of the form $A_2 \sqsubseteq \neg A_3$ or $A_3 \sqsubseteq \neg A_2$, where A_3 is an atomic concept, (or $A_2 \sqsubseteq \neg \exists Q$ or $\exists Q \sqsubseteq \neg A_2$, where Q is a basic role) such that $A_3(a) \in chase_i(\mathcal{O})$ (resp., there exists a constant a' such that $Q(a, a') \in chase_i(\mathcal{O})$). Notice that this means that $a \in A_3^{can_i(\mathcal{O})}$ (resp., $a \in \exists Q^{can_i(\mathcal{O})}$). It is easy to see that, if such a NI exists, then also $A_1 \sqsubseteq \neg A_3$ (resp., $A_1 \sqsubseteq \neg \exists Q$) belongs to $cln(\mathcal{T})$, according to NI-closure rule 3 in Definition 4.7. Since $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{A_2(a)\}$, then $A_1 \sqsubseteq \neg A_3$ (resp., $A_1 \sqsubseteq \neg \exists Q$) is not satisfied already by $can_i(\mathcal{O})$, if $A_3 \neq A_2$. If $A_3 = A_2$, we need to consider again NI-closure rule 3, according to which, from the fact that $A_1 \sqsubseteq A_2$ in $\mathcal{T}_p$, and $A_1 \sqsubseteq \neg A_2$ in $cln(\mathcal{T})$, it follows that $A_1 \sqsubseteq \neg A_1$ is in $cln(\mathcal{T})$, and therefore $A_1(a)$ is not satisfied already by $can_i(\mathcal{O})$. In both cases, we have thus contradicted the assumption that $can_i(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A}\rangle$. With an almost identical argument we can prove the inductive step also in those cases in which $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by executing rule **cr3** or rule **cr5** (in this last case, in particular, we need to use in the proof NI-closure rules 4, 5 and 6). As for the cases in which $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by applying rule **cr2**, we proceed as follows (for rule **cr4** the proof is analogous). According to **cr2**, a PI of the form $A \sqsubseteq \exists Q$, where A is an atomic concept in $\mathcal{T}$, and Q is a basic role in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion $A(a)$ such that there does not exist $a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{Q(a, a_2)\}$, where a_2 follows lexicographically all constants appearing in $chase_i(\mathcal{O})$ (notice that this means that $a \in \exists Q^{can_{i+1}(\mathcal{O})}$). Now, if $can_{i+1}(\mathcal{O})$ is not a model of $cln(\mathcal{T})$, there must exist a NI in $cln(\mathcal{T})$ of the form $\exists Q \sqsubseteq \neg B$, where B is a basic concept, or of the form $\exists Q^- \sqsubseteq \neg \exists Q^-$, or of the form $Q \sqsubseteq \neg Q$. As for the first form of NI, we can reach a contradiction as done above for the case of execution of chase rule **cr1**. As for the last two forms of NIs, according to NI-closure rule 7, we have that if (at least) one of these NIs is in $cln(\mathcal{T})$, then also $\exists Q \sqsubseteq \neg \exists Q$ is in $cln(\mathcal{T})$, and thus we can again reason on a NI of the first form to reach a contradiction. □

The following corollary is an interesting consequence of the lemma above.

Corollary 4.10. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and α a negative inclusion assertion or a functionality assertion. We have that, if $\mathcal{T} \models \alpha$, then $cln(\mathcal{T}) \models \alpha$.*

Proof. We first consider the case in which α is a NI. We prove the claim by contradiction. Let us assume that $\mathcal{T} \models \alpha$ and $cln(\mathcal{T}) \not\models \alpha$. We show that from $cln(\mathcal{T}) \not\models \alpha$ one can construct a model of $\mathcal{T}$ which does not satisfy α, thus obtaining a contradiction.

Let us assume that $\alpha = A_1 \sqsubseteq \neg A_2$, where A_1 and A_2 are atomic concepts in $\mathcal{T}$, and consider the *DL-Lite$_\mathcal{A}$* ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{A} = \{A_1(a), A_2(a)\}$. We show that $can(\mathcal{O})$ is the model we are looking for, i.e., $can(\mathcal{O}) \models \mathcal{T}$ but $can(\mathcal{O}) \not\models \alpha$. The last property follows trivially by the form of $\mathcal{A}$. Hence, in the following we concentrate on proving that $can(\mathcal{O}) \models \mathcal{T}$.

We recall that $DB(\mathcal{A})$ is such that $A_1^{DB(\mathcal{A})} = \{a\}$, $A_2^{DB(\mathcal{A})} = \{a\}$, $A^{DB(\mathcal{A})} = \emptyset$ for each atomic concept $A \in \mathcal{T}$ such that $A \neq A_1$ and $A \neq A_2$, and $P^{DB(\mathcal{A})} = \emptyset$ for each atomic role $P \in \mathcal{T}$. Therefore, the only NIs that can be violated by $DB(\mathcal{A})$ are $A_1 \sqsubseteq \neg A_2$, $A_2 \sqsubseteq \neg A_1$, $A_1 \sqsubseteq \neg A_1$, and $A_2 \sqsubseteq \neg A_2$. By assumption, we have that $cln(\mathcal{T}) \not\models$

$A_1 \sqsubseteq \neg A_2$, and therefore also $cln(\mathcal{T}) \not\models A_2 \sqsubseteq \neg A_1$. From this, it follows also that $cln(\mathcal{T}) \not\models A_1 \sqsubseteq \neg A_1$ and $cln(\mathcal{T}) \not\models A_2 \sqsubseteq \neg A_2$, since either $A_1 \sqsubseteq \neg A_1$ or $A_2 \sqsubseteq \neg A_2$ logically implies $A_1 \sqsubseteq \neg A_2$. Moreover, being $\mathcal{A} = \{A_1(a), A_2(a)\}$, $DB(\mathcal{A})$ cannot violate functionality assertions. Therefore, we can conclude that $DB(\mathcal{A}) \models cln(\mathcal{T})$ and hence $DB(\mathcal{A}) \models \langle cln(\mathcal{T}), \mathcal{A}\rangle$. Then, from Lemma 4.9 it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$.

Proceeding analogously as done above, we can easily prove the claim in those cases in which α is one of $A \sqsubseteq \neg\exists Q$, $\exists Q \sqsubseteq \neg A$, $\exists Q_1 \sqsubseteq \neg\exists Q_2$, or $Q_1 \sqsubseteq \neg Q_2$.

The proof for the case in which α is a functionality assertion of the form $(\mathsf{funct}\ Q)$ can be obtained in an analogous way, by constructing the canonical interpretation starting from an ABox with the assertions $Q(a, a_1)$ and $Q(a, a_2)$. $\square$

4.3 FOL-Rewritability of Ontology Satisfiability

Before providing the main results of this subsection, we need also the following crucial property, which asserts that to establish satisfiability of an ontology, we can resort to constructing the canonical interpretation.

Lemma 4.11. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_{\mathcal{A}}$ ontology. Then, $can(\mathcal{O})$ is a model of $\mathcal{O}$ if and only if $\mathcal{O}$ is satisfiable.*

Proof. "$\Rightarrow$" If $can(\mathcal{O})$ is a model of $\mathcal{O}$, then $\mathcal{O}$ is obviously satisfiable.

"$\Leftarrow$" We prove this direction by showing that if $can(\mathcal{O})$ is not a model of $\mathcal{O}$, then $\mathcal{O}$ is unsatisfiable. By Lemma 4.9 ("if" direction), it follows that $DB(\mathcal{A})$ is not a model of $\langle cln(\mathcal{T}), \mathcal{A}\rangle$, and since $DB(\mathcal{A}) \models \mathcal{A}$ by construction, $DB(\mathcal{A}) \not\models cln(\mathcal{T})$. This means that there exists a NI or functionality assertion α such that $DB(\mathcal{A}) \not\models \alpha$ and $cln(\mathcal{T}) \models \alpha$, and hence by Lemma 4.8 $\mathcal{T} \not\models \alpha$.

Consider the case where α if of the form $B_1 \sqsubseteq \neg B_2$, where B_1 and B_2 are basic concepts (resp., α is of the form $Q_1 \sqsubseteq \neg Q_2$, where Q_1 and Q_2 are basic roles). Then, there exists $a_1 \in \Delta^{DB(\mathcal{A})}$ such that $a_1 \in B_1^{DB(\mathcal{A})}$ and $a_1 \in B_2^{DB(\mathcal{A})}$ (resp., there exist $a_1, a_2 \in \Delta^{DB(\mathcal{A})}$ such that $(a_1, a_2) \in Q_1^{DB(\mathcal{A})}$ and $(a_1, a_2) \in Q_2^{DB(\mathcal{A})}$). Let us assume by contradiction that a model $\mathcal{M} = \langle \Delta^{\mathcal{M}}, \cdot^{\mathcal{M}}\rangle$ of $\mathcal{O}$ exists. For each model $\mathcal{M}$, we can construct a homomorphism ψ from $\Delta^{DB(\mathcal{A})}$ to $\Delta^{\mathcal{M}}$ such that $\psi(a) = a^{\mathcal{M}}$ for each constant a occurring in $\mathcal{A}$ (notice that $\mathcal{M}$ assigns a distinct object to each such constant, since $\mathcal{M} \models \mathcal{A}$). From the fact that $\mathcal{M}$ satisfies the membership assertions in $\mathcal{A}$, it easily follows that $\psi(a_1) \in B_1^{\mathcal{M}}$ and $\psi(a_1) \in B_2^{\mathcal{M}}$ (resp., $(\psi(a_1), \psi(a_2)) \in Q_1^{\mathcal{M}}$ and $(\psi(a_1), \psi(a_2)) \in Q_2^{\mathcal{M}}$). But this makes the NI $B_1 \sqsubseteq \neg B_2$ (resp., $Q_1 \sqsubseteq \neg Q_2$) be violated also in $\mathcal{M}$, and since a model cannot violate a NI that is logically implied by $\mathcal{T}$, it contradicts the fact that $\mathcal{M}$ is a model of $\mathcal{O}$.

The proof for the case where α is a functionality assertion of the form $(\mathsf{funct}\ Q)$ can be obtained in an analogous way, considering that there exist $a_1, a_2, a_3 \in \Delta^{DB(\mathcal{A})}$ such that $(a_1, a_2) \in Q^{DB(\mathcal{A})}$ and $(a_1, a_3) \in Q^{DB(\mathcal{A})}$. $\square$

Notice that, the construction of $can(\mathcal{O})$ is in general neither convenient nor possible, since $can(\mathcal{O})$ may be infinite. However, by simply combining Lemma 4.9 and Lemma 4.11, we obtain the notable result that to check satisfiability of an ontology, it

```
Algorithm Satisfiable($\mathcal{O}$)
Input: DL-Lite$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: true if $\mathcal{O}$ is satisfiable, false otherwise
begin
  $q_{unsat(\mathcal{T})} := \{\bot\}$;
  for each $\alpha \in cln(\mathcal{T})$ do $q_{unsat(\mathcal{T})} := q_{unsat(\mathcal{T})} \cup \{\delta(\alpha)\}$;
  if $q_{unsat(\mathcal{T})}^{DB(\mathcal{A})} = \emptyset$ then return true; else return false;
end
```

Fig. 9. The algorithm Satisfiable that checks satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology

is sufficient (and necessary) to look at $DB(\mathcal{A})$ (provided we have computed $cln(\mathcal{T})$). More precisely, the next theorem shows that a contradiction on a *DL-Lite*$_\mathcal{A}$ ontology may hold only if a membership assertion in the ABox contradicts a functionality assertion or a NI implied by the closure $cln(\mathcal{T})$.

Theorem 4.12. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology. Then, $\mathcal{O}$ is satisfiable if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

Proof. "$\Rightarrow$" If $\mathcal{O}$ is satisfiable, from Lemma 4.11 ("only-if" direction), it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore, from Lemma 4.9 ("only-if" direction), it follows that $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.

"$\Leftarrow$" If $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$, from Lemma 4.9 ("if" direction), it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore $\mathcal{O}$ is satisfiable. □

At this point, it is not difficult to show that verifying whether $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$ can be done by simply evaluating a suitable boolean FOL query, in fact a boolean UCQ with inequalities, over $DB(\mathcal{A})$ itself. In particular we define a translation function δ from assertions in $cln(\mathcal{T})$ to boolean CQs with inequalities, as follows:

$$\begin{aligned}
\delta((\mathsf{funct}\ P)) &= \exists x, y_1, y_2.\, P(x, y_1) \wedge P(x, y_2) \wedge y_1 \neq y_2 \\
\delta((\mathsf{funct}\ P^-)) &= \exists x_1, x_2, y.\, P(x_1, y) \wedge P(x_2, y) \wedge x_1 \neq x_2 \\
\delta(B_1 \sqsubseteq \neg B_2) &= \exists x.\, \gamma_1(B_1, x) \wedge \gamma_2(B_2, x) \\
\delta(Q_1 \sqsubseteq \neg Q_2) &= \exists x, y.\, \rho(Q_1, x, y) \wedge \rho(Q_2, x, y)
\end{aligned}$$

where in the last two equations

$$\gamma_i(B, x) = \begin{cases} A(x), & \text{if } B = A, \\ \exists y_i.\, P(x, y_i), & \text{if } B = \exists P, \\ \exists y_i.\, P(y_i, x), & \text{if } B = \exists P^-, \end{cases} \qquad \rho(Q, x, y) = \begin{cases} P(x, y), & \text{if } Q = P, \\ P(y, x), & \text{if } Q = P^-. \end{cases}$$

The algorithm Satisfiable, shown in Figure 9, takes as input a *DL-Lite*$_\mathcal{A}$ ontology, computes $DB(\mathcal{A})$ and $cln(\mathcal{T})$, and evaluates over $DB(\mathcal{A})$ the boolean FOL query $q_{unsat(\mathcal{T})}$ obtained by taking the union of all FOL formulas returned by the application of the above function δ to every assertion in $cln(\mathcal{T})$. In the algorithm, the symbol $\bot$ indicates a predicate whose evaluation is *false* in every interpretation. Notice that in the case in which neither functionality assertions nor negative inclusion assertions occur in $\mathcal{T}$, $q_{unsat(\mathcal{T})} = \bot$, and therefore $q_{unsat(\mathcal{T})}^{DB(\mathcal{A})} = \bot^{DB(\mathcal{A})} = \emptyset$ and Satisfiable($\mathcal{O}$) returns *true*.

TBox $\mathcal{T}'_{fbc}$					
$League$	$\sqsubseteq$	$\exists OF$	$Match$	$\sqsubseteq$	$\exists HOME$
$\exists OF$	$\sqsubseteq$	$League$	$\exists HOME$	$\sqsubseteq$	$Match$
$\exists OF^-$	$\sqsubseteq$	$Nation$	$\exists HOME^-$	$\sqsubseteq$	$Team$
$Round$	$\sqsubseteq$	$\exists BELONGS\text{-}TO$	$Match$	$\sqsubseteq$	$\exists HOST$
$\exists BELONGS\text{-}TO$	$\sqsubseteq$	$Round$	$\exists HOST$	$\sqsubseteq$	$Match$
$\exists BELONGS\text{-}TO^-$	$\sqsubseteq$	$League$	$\exists HOST^-$	$\sqsubseteq$	$Team$
$Match$	$\sqsubseteq$	$\exists PLAYED\text{-}IN$	$Match$	$\sqsubseteq$	$\neg Round$
$\exists PLAYED\text{-}IN$	$\sqsubseteq$	$Match$			
$\exists PLAYED\text{-}IN^-$	$\sqsubseteq$	$Round$	$(\mathsf{funct}\ OF)$		
$PlayedMatch$	$\sqsubseteq$	$Match$	$(\mathsf{funct}\ BELONGS\text{-}TO)$		
$ScheduledMatch$	$\sqsubseteq$	$Match$	$(\mathsf{funct}\ HOME)$		
$PlayedMatch$	$\sqsubseteq$	$\neg ScheduledMatch$	$(\mathsf{funct}\ HOST)$		

ABox $\mathcal{A}'_{fbc}$		
$League(it2009)$		$PLAYED\text{-}IN(m7RJ, r7)$
$Round(r7)$	$BELONGS\text{-}TO(r7, it2009)$	$PLAYED\text{-}IN(m8NT, r8)$
$Round(r8)$	$BELONGS\text{-}TO(r8, it2009)$	$PLAYED\text{-}IN(m8RM, r8)$
$PlayedMatch(m7RJ)$	$HOME(m7RJ, roma)$	$HOST(m7RJ, juventus)$
$Match(m8NT)$	$HOME(m8NT, napoli)$	$HOST(m8NT, torino)$
$Match(m8RM)$	$HOME(m8RM, roma)$	$HOST(m8RM, milan)$
$Team(roma)$	$Team(napoli)$	$Team(juventus)$

Fig. 10. The simplified *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O}'_{fbc} = \langle \mathcal{T}'_{fbc}, \mathcal{A}'_{fbc} \rangle$ for the football championship example

Lemma 4.13. *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A}}$ ontology. Then, the algorithm* $\mathsf{Satisfiable}(\mathcal{O})$ *terminates, and $\mathcal{O}$ is satisfiable if and only if* $\mathsf{Satisfiable}(\mathcal{O}) = true$.

Proof. Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. Since $cln(\mathcal{T})$ is a finite set of membership and functionality assertions, the algorithm terminates. By Theorem 4.12, we have that $DB(\mathcal{A})$ is a model of all assertions in $cln(\mathcal{T})$ if and only if $\mathcal{O}$ is satisfiable. The query $q_{unsat(\mathcal{T})}$ verifies whether there exists an assertion α that is violated in $DB(\mathcal{A})$, by expressing its negation as a FOL formula $\delta(\alpha)$ and evaluating it over $DB(\mathcal{A})$. □

As a direct consequence of Lemma 4.13, we get:

Theorem 4.14. *In DL-Lite$_{\mathcal{A}}$, ontology satisfiability is FOL-rewritable.*

Example 4.15. We refer again to Example 2.3 about the football championship domain. Let $\mathcal{O}'_{fbc} = \langle \mathcal{T}'_{fbc}, \mathcal{A}'_{fbc} \rangle$ be the *DL-Lite*$_{\mathcal{A}}$ ontology shown in Figure 10, which is a simplified version of the ontology $\mathcal{O}_{fbc}$ used in Example 2.3. Specifically, the TBox $\mathcal{T}'_{fbc}$ is obtained from the TBox $\mathcal{T}_{fbc}$ shown in Figure 3 by ignoring identification constraints, all assertions involving value domains or attributes, and the functionality assertion on *PLAYED-IN*. The ABox $\mathcal{A}'_{fbc}$ is obtained from the ABox $\mathcal{A}_{fbc}$ shown in Figure 4 by considering only the membership assertions involving concepts and roles (and ignoring those for attributes). To check satisfiability of $\mathcal{O}'_{fbc}$, we first compute $cln(\mathcal{T}'_{fbc})$, which, besides the functionality assertions shown in Figure 10 contains the NIs shown in Figure 11.

$$\begin{array}{rlrl}
\mathit{PlayedMatch} & \sqsubseteq \neg \mathit{ScheduledMatch} & & \\
\mathit{Match} & \sqsubseteq \neg \mathit{Round} & \exists \mathit{PLAYED\text{-}IN} & \sqsubseteq \neg \mathit{Round} \\
\mathit{PlayedMatch} & \sqsubseteq \neg \mathit{Round} & \exists \mathit{HOME} & \sqsubseteq \neg \mathit{Round} \\
\mathit{ScheduledMatch} & \sqsubseteq \neg \mathit{Round} & \exists \mathit{HOST} & \sqsubseteq \neg \mathit{Round} \\
\mathit{Match} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- & \exists \mathit{PLAYED\text{-}IN} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- \\
\mathit{PlayedMatch} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- & \exists \mathit{HOME} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- \\
\mathit{ScheduledMatch} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- & \exists \mathit{HOST} & \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- \\
\mathit{Match} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} & \exists \mathit{PLAYED\text{-}IN} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} \\
\mathit{PlayedMatch} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} & \exists \mathit{HOME} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} \\
\mathit{ScheduledMatch} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} & \exists \mathit{HOST} & \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}
\end{array}$$

Fig. 11. The negative inclusions in $cln(\mathcal{T}'_{fbc})$

We show some of the boolean queries obtained by applying the translation function δ to the NIs in Figure 11:

$$\begin{aligned}
\delta(\mathit{PlayedMatch} \sqsubseteq \neg \mathit{ScheduledMatch}) &= \exists x. \mathit{PlayedMatch}(x) \wedge \mathit{ScheduledMatch}(x) \\
\delta(\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \neg \mathit{Round}) &= \exists x. (\exists y. \mathit{PLAYED\text{-}IN}(x,y)) \wedge \mathit{Round}(x) \\
\delta(\mathit{Match} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-) &= \exists x. \mathit{Match}(x) \wedge (\exists y. \mathit{PLAYED\text{-}IN}(y,x)) \\
\delta(\exists \mathit{HOME} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-) &= \exists x. (\exists y_1. \mathit{HOME}(x,y_1)) \wedge (\exists y_2. \mathit{PLAYED\text{-}IN}(y_2,x))
\end{aligned}$$

We also show one of the boolean queries obtained by applying the translation function δ to the functionality assertions in $\mathcal{T}'_{fbc}$ (the other queries are defined analogously):

$$\delta((\mathsf{funct}\ \mathit{OF})) = \exists x, y_1, y_2. \mathit{OF}(x,y_1) \wedge \mathit{OF}(x,y_2) \wedge y_1 \neq y_2.$$

The union of the boolean queries for all the NIs and for all the functionality assertions is $q_{unsat(\mathcal{T}'_{fbc})}$. Such a query, when evaluated over $DB(\mathcal{A}'_{fbc})$, returns $false$, thus showing that $\mathcal{O}'_{fbc}$ is satisfiable.

As a further example, consider now the TBox $\mathcal{T}''_{fbc}$ obtained from $\mathcal{T}'_{fbc}$ by introducing a new role *NEXT* and adding the role inclusion assertion $\mathit{NEXT} \sqsubseteq \mathit{PLAYED\text{-}IN}$. In this case $cln(\mathcal{T}''_{fbc})$ consists of $cln(\mathcal{T}'_{fbc})$ plus the following NIs:

$$\begin{array}{lll}
\exists \mathit{NEXT} \sqsubseteq \neg \mathit{Round} & \exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{Match} & \exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN} \\
\exists \mathit{NEXT} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^- & \exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{PlayedMatch} & \exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{HOME} \\
\exists \mathit{NEXT} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO} & \exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{ScheduledMatch} & \exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{HOST}
\end{array}$$

So $q_{unsat(\mathcal{T}''_{fbc})}$ includes the disjuncts of $q_{unsat(\mathcal{T}'_{fbc})}$ plus those obtained from the above NIs. Since $q_{unsat(\mathcal{T}''_{fbc})}$, when evaluated over $DB(\mathcal{A}'_{fbc})$, returns $false$, we conclude that $\mathcal{O}''_{fbc} = \langle \mathcal{T}''_{fbc}, \mathcal{A}'_{fbc} \rangle$ is satisfiable.

If we instead add to $\mathcal{T}'_{fbc}$ the functionality assertion $(\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)$, we obtain a TBox $\mathcal{T}'''_{fbc}$ whose NI-closure is $cln(\mathcal{T}'''_{fbc}) = cln(\mathcal{T}'_{fbc}) \cup \{(\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)\}$. In this case, $q_{unsat(\mathcal{T}'''_{fbc})}$ includes the disjuncts of $q_{unsat(\mathcal{T}'_{fbc})}$ plus the query

$$\delta((\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)) = \exists x_1, x_2, y. \mathit{PLAYED\text{-}IN}(x_1,y) \wedge \mathit{PLAYED\text{-}IN}(x_2,y) \wedge x_1 \neq x_2.$$

Now, $q_{unsat(\mathcal{T}'''_{fbc})}^{DB(\mathcal{A}'_{fbc})}$ is $true$, and hence $\mathcal{O}'''_{fbc} = \langle \mathcal{T}'''_{fbc}, \mathcal{A}'_{fbc} \rangle$ is unsatisfiable. ■

4.4 Concept and Role Satisfiability and Logical Implication

We start by showing that concept and role satisfiability with respect to a TBox (or an ontology) can be reduced to ontology satisfiability.

Theorem 4.16. *Let $\mathcal{T}$ be a DL-Lite$_{\mathcal{A}}$ TBox, C a general concept, and Q a basic role. Then the following holds:*

(1) C is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology

$$\mathcal{O}_{\mathcal{T},C} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C\},\ \{A_{new}(a)\}\rangle$$

is satisfiable, where A_{new} is an atomic concept not appearing in $\mathcal{T}$, and a is a fresh constant.

(2) Q is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology

$$\mathcal{O}_{\mathcal{T},Q} = \langle \mathcal{T},\ \{Q(a_1, a_2)\}\rangle$$

is satisfiable, where a_1 and a_2 are two fresh constants.

(3) $\neg Q$ is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology

$$\mathcal{O}_{\mathcal{T},\neg Q} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q\},\ \{P_{new}(a_1, a_2)\}\rangle$$

is satisfiable, where P_{new} is an atomic role not appearing in $\mathcal{T}$, and a_1 and a_2 are two fresh constants.

Proof. We observe that for roles we have distinguished Case 2 from Case 3, since we need to ensure that the ontology that we obtain in the reduction is a valid *DL-Lite*$_{\mathcal{A}}$ ontology, and hence does not introduce a positive role inclusion assertion on a possibly functional role. We then give only the proof for Case 1, i.e., for concepts, since Case 2 is immediate, and the proof for Case 3 is analogous to that for Case 1.

"$\Leftarrow$" If $\mathcal{O}_{\mathcal{T},C}$ is satisfiable, there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $A_{new}^{\mathcal{M}} \subseteq C^{\mathcal{M}}$ and $a^{\mathcal{M}} \in A_{new}^{\mathcal{M}}$. Hence $C^{\mathcal{M}} \neq \emptyset$, and C is satisfiable w.r.t. $\mathcal{T}$.

"$\Rightarrow$" If C is satisfiable w.r.t. $\mathcal{T}$, there exists a model $\mathcal{M}$ of $\mathcal{T}$ and an object $o \in \Delta^{\mathcal{M}}$ such that $o \in C^{\mathcal{M}}$. We can extend $\mathcal{M}$ by defining $a^{\mathcal{I}} = o$ and $A_{new}^{\mathcal{I}} = \{o\}$, and obtain a model of $\mathcal{O}_{\mathcal{T},C}$. □

Next, we show that both instance checking and subsumption can be reduced to ontology satisfiability. We first consider the problem of instance checking for concept expressions.

Theorem 4.17. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_{\mathcal{A}}$ ontology, C a general concept, and a a constant appearing in $\mathcal{O}$. Then $\mathcal{O} \models C(a)$ if and only if the ontology*

$$\mathcal{O}_{C(a)} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq \neg C\},\ \mathcal{A} \cup \{A_{new}(a)\}\rangle$$

is unsatisfiable, where A_{new} is an atomic concept not appearing in $\mathcal{O}$.

Proof. "⇒" Suppose that $\mathcal{O} \models C(a)$, but there exists a model $\mathcal{M}'$ of $\mathcal{O}_{C(a)}$. Then $\mathcal{M}' \models A_{new}(a)$ and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C$. But then $\mathcal{M}' \models \neg C(a)$. Observe that $\mathcal{M}'$ is a model of $\mathcal{O}$, hence we get a contradiction.

"⇐" Suppose that $\mathcal{O}_{C(a)}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{O}$ such that $\mathcal{M} \models \neg C(d)$. Then we can define an interpretation $\mathcal{M}'$ of $\mathcal{O}_{C(a)}$ that interprets all constants, concepts, and roles in $\mathcal{O}$ as before, and assigns to A_{new} (which does not appear in $\mathcal{O}$) the extension $A_{new}^{\mathcal{M}'} = \{a^{\mathcal{M}}\}$. Now, $\mathcal{M}'$ is still a model of $\mathcal{O}$, and moreover we have that $\mathcal{M}' \models A_{new}(a)$ and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C$, hence $\mathcal{M}'$ is a model of $\mathcal{O}_{C(a)}$. Thus we get a contradiction. □

The analogous of the above theorem holds for the problem of instance checking for role expressions.

Theorem 4.18. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A}}$ ontology, Q a basic role, and a_1 and a_2 two constants appearing in $\mathcal{O}$. Then*

(1) $\mathcal{O} \models Q(a_1, a_2)$ if and only if the ontology

$$\mathcal{O}_{Q(a_1,a_2)} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q\},\ \mathcal{A} \cup \{P_{new}(a_1, a_2)\} \rangle$$

is unsatisfiable, where P_{new} is an atomic role not appearing in $\mathcal{O}$.

(2) $\mathcal{O} \models \neg Q(a_1, a_2)$ if and only if the ontology

$$\mathcal{O}_{\neg Q(a_1,a_2)} = \langle \mathcal{T},\ \mathcal{A} \cup \{Q(a_1, a_2)\} \rangle$$

is unsatisfiable.

Proof. We observe that we need again to distinguish the two cases to ensure that the ontology that we obtain in the reduction is a valid *DL-Lite$_{\mathcal{A}}$* ontology. Then, the proof of Case 1 is similar to the proof of Theorem 4.17, while Case 2 is obvious. □

We now address the subsumption problem and provide different reductions of this problem to the problem of ontology satisfiability. The case of subsumption between concepts is dealt with by the following theorem, and the case of subsumption between roles, is considered in the two subsequent theorems.

Theorem 4.19. *Let $\mathcal{T}$ be a DL-Lite$_{\mathcal{A}}$ TBox, and C_1 and C_2 two general concepts. Then, $\mathcal{T} \models C_1 \sqsubseteq C_2$ if and only if the ontology*

$$\mathcal{O}_{C_1 \sqsubseteq C_2} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C_1, A_{new} \sqsubseteq \neg C_2\},\ \{A_{new}(a)\} \rangle,$$

is unsatisfiable, where A_{new} is an atomic concept not appearing in $\mathcal{T}$, and a is a fresh constant.

Proof. "⇒" Suppose that $\mathcal{T} \models C_1 \sqsubseteq C_2$, but there exists a model $\mathcal{M}'$ of $\mathcal{O}_{C_1 \sqsubseteq C_2}$. Then $\mathcal{M}' \models A_{new}(a)$, $\mathcal{M}' \models A_{new} \sqsubseteq C_1$, and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C_2$. But then $\mathcal{M}' \models C_1(a)$ and $\mathcal{M}' \models \neg C_2(a)$. Observe that $\mathcal{M}'$ is a model of $\mathcal{T}$, hence we get a contradiction.

"⇐" Suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $o \in C_1^{\mathcal{M}}$ and $o \notin C_2^{\mathcal{M}}$ for some object o in the domain of $\mathcal{M}$. Then we can define an interpretation $\mathcal{M}'$ of $\mathcal{O}_{C_1 \sqsubseteq C_2}$ that interprets all concepts and roles in $\mathcal{T}$ as before, and assigns to a the extensions $a^{\mathcal{M}'} = o$, and to A_{new} (which does not appear in $\mathcal{T}$) the extension $A_{new}^{\mathcal{M}'} = \{o\}$. Now, $\mathcal{M}'$ is still a model of $\mathcal{T}$, and moreover we have that $\mathcal{M}' \models A_{new}(a)$, $\mathcal{M}' \models A_{new} \sqsubseteq C_1$, and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C_2$. Hence $\mathcal{M}'$ is a model of $\mathcal{O}_{C_1 \sqsubseteq C_2}$, and we get a contradiction. □

Theorem 4.20. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and Q_1 and Q_2 two basic roles. Then,*

(1) $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ if and only if the ontology

$$\mathcal{O}_{Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_2\},\ \{Q_1(a_1, a_2), P_{new}(a_1, a_2)\}\rangle$$

is unsatisfiable, where P_{new} is an atomic role not appearing in $\mathcal{T}$, and a_1, a_2 are two fresh constants.

(2) $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ if and only if the ontology

$$\mathcal{O}_{\neg Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_1, P_{new} \sqsubseteq \neg Q_2\},\ \{P_{new}(a_1, a_2)\}\rangle$$

is unsatisfiable, where P_{new} is an atomic role not appearing in $\mathcal{T}$, and a_1, a_2 are two fresh constants.

(3) $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ if and only if the ontology

$$\mathcal{O}_{Q_1 \sqsubseteq \neg Q_2} = \langle \mathcal{T},\ \{Q_1(a_1, a_2), Q_2(a_1, a_2)\}\rangle$$

is unsatisfiable, where a_1, a_2 are two fresh constants.

(4) $\mathcal{T} \models \neg Q_1 \sqsubseteq \neg Q_2$ if and only if the ontology

$$\mathcal{O}_{\neg Q_1 \sqsubseteq \neg Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_1\},\ \{Q_2(a_1, a_2), P_{new}(a_1, a_2)\}\rangle$$

is unsatisfiable, where P_{new} is an atomic role not appearing in $\mathcal{T}$, and a_1, a_2 are two fresh constants.

Proof. Let R_i, for $i \in \{1, 2\}$, denote either Q_i or $\neg Q_i$, depending on the case we are considering. First of all, we observe that in all four cases, the ontology $\mathcal{O}_{R_1 \sqsubseteq R_2}$ constructed in the reduction is a valid *DL-Lite$_\mathcal{A}$* ontology.

"$\Rightarrow$" Suppose that $\mathcal{T} \models R_1 \sqsubseteq R_2$, but there exists a model $\mathcal{M}$ of $\mathcal{O}_{R_1 \sqsubseteq R_2}$. In Case 1, since $\mathcal{M} \models Q_1(a_1, a_2)$ and $\mathcal{M} \models P_{new}(a_1, a_2)$, and since $P_{new}^{\mathcal{M}} \subseteq (\neg Q_2)^{\mathcal{M}}$, we have that $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \in Q_1^{\mathcal{M}}$ and $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_2^{\mathcal{M}}$. Since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. In Case 2, since $\mathcal{M} \models P_{new}(a_1, a_2)$, and since $P_{new}^{\mathcal{M}} \subseteq (\neg Q_1)^{\mathcal{M}}$ and $P_{new}^{\mathcal{M}} \subseteq (\neg Q_2)^{\mathcal{M}}$, we have that $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_1^{\mathcal{M}}$ and $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_2^{\mathcal{M}}$. Since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. In Case 3, since $\mathcal{M} \models Q_1(a_1, a_2)$ and $\mathcal{M} \models Q_2(a_1, a_2)$, and since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. Case 4 is analogous to Case 1, since $\mathcal{T} \models \neg Q_1 \sqsubseteq \neg Q_2$ iff $\mathcal{T} \models Q_2 \sqsubseteq Q_1$.

"$\Leftarrow$" Suppose that $\mathcal{O}_{R_1 \sqsubseteq R_2}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $(o_a, o_b) \in R_1^{\mathcal{M}}$ and $(o_a, o_b) \notin R_2^{\mathcal{M}}$ for some pair of objects in the domain of $\mathcal{M}$. We first show that we can assume w.l.o.g. that o_a and o_b are distinct objects. Indeed, if $o_a = o_b$, we can construct a new model $\mathcal{M}_d$ of $\mathcal{T}$ as follows:

$\Delta^{\mathcal{M}_d} = \Delta^{\mathcal{M}} \times \{1,2\}$, $A^{\mathcal{M}_d} = A^{\mathcal{M}} \times \{1,2\}$ for each atomic concept A, and $P^{\mathcal{M}_d} = (\{((o,1),(o',1)),((o,2),(o',2)) \mid (o,o') \in P^{\mathcal{M}}\} \cup U) \setminus V$, where

$$U = \begin{cases} \emptyset, & \text{if } (o_a,o_a) \notin P^{\mathcal{M}} \\ \{((o_a,1),(o_a,2)),((o_a,2),(o_a,1))\}, & \text{if } (o_a,o_a) \in P^{\mathcal{M}} \end{cases}$$
$$V = \begin{cases} \emptyset, & \text{if } (o_a,o_a) \notin P^{\mathcal{M}} \\ \{((o_a,1),(o_a,1)),((o_a,2),(o_a,2))\}, & \text{if } (o_a,o_a) \in P^{\mathcal{M}} \end{cases}$$

for each atomic role P. It is immediate to see that $\mathcal{M}_d$ is still a model of $\mathcal{T}$ containing a pair of distinct objects in $R_1^{\mathcal{M}_d}$ and not in $R_2^{\mathcal{M}_d}$.

Now, given that we can assume that $o_a \neq o_b$, we can define an interpretation $\mathcal{M}'$ of $\mathcal{O}_{R_1 \sqsubseteq R_2}$ that interprets all concepts and roles in $\mathcal{T}$ as before, and assigns to a_1 and a_2 respectively the extensions $a_1^{\mathcal{M}'} = o_a$ and $a_2^{\mathcal{M}'} = o_b$, and to P_{new} (which does not appear in $\mathcal{T}$), if present in $\mathcal{O}_{R_1 \sqsubseteq R_2}$, the extension $P_{new}^{\mathcal{M}'} = \{(o_a,o_b)\}$. We have that $\mathcal{M}'$ is still a model of $\mathcal{T}$, and moreover it is easy to see that in all four cases $\mathcal{M}'$ is a model of $\mathcal{O}_{R_1 \sqsubseteq R_2}$. Thus we get a contradiction. □

We remark that in the previous theorem the answer in Case 2 will always be false, since a *DL-Lite*$_{\mathcal{A}}$ TBox cannot imply an inclusion with a negated role on the left-hand side. We have nevertheless included this case in the theorem statement to cover explicitly all possibilities.

The following theorem characterizes logical implication of a functionality assertion in *DL-Lite*$_{\mathcal{A}}$, in terms of subsumption between roles.

Theorem 4.21. *Let $\mathcal{T}$ be a DL-Lite$_{\mathcal{A}}$ TBox and Q a basic role. Then, $\mathcal{T} \models (\mathsf{funct}\ Q)$ if and only if either $(\mathsf{funct}\ Q) \in \mathcal{T}$ or $\mathcal{T} \models Q \sqsubseteq \neg Q$.*

Proof. "$\Leftarrow$" The case in which $(\mathsf{funct}\ Q) \in \mathcal{T}$ is trivial. Instead, if $\mathcal{T} \models Q \sqsubseteq \neg Q$, then $Q^{\mathcal{I}} = \emptyset$ and hence $\mathcal{I} \models (\mathsf{funct}\ Q)$, for every model $\mathcal{I}$ of $\mathcal{T}$.

"$\Rightarrow$" We assume that neither $(\mathsf{funct}\ Q) \in \mathcal{T}$ nor $\mathcal{T} \models Q \sqsubseteq \neg Q$, and we construct a model of $\mathcal{T}$ that is not a model of $(\mathsf{funct}\ Q)$. First of all, notice that, since $\mathcal{T}$ does not imply $Q \sqsubseteq \neg Q$, it also does not imply $\exists Q \sqsubseteq \neg \exists Q$ and $\exists Q^- \sqsubseteq \neg \exists Q^-$. Now, consider the ABox $\mathcal{A} = \{Q(a,a_1), Q(a,a_2)\}$, where a, a_1, and a_2 are pairwise distinct objects, and the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. According to Theorem 4.12, $\mathcal{O}$ is satisfiable if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$. Since $Q(a,a_1)$ and $Q(a,a_2)$ are the only membership assertions in $\mathcal{A}$, the only assertions that $DB(\mathcal{A})$ can violate are (*i*) the NIs $Q \sqsubseteq \neg Q$, $\exists Q \sqsubseteq \neg \exists Q$, and $\exists Q^- \sqsubseteq \neg \exists Q^-$, and (*ii*) the functionality assertion $(\mathsf{funct}\ Q)$. But, by assumption, $\mathcal{T}$ does not imply any of such assertions, and therefore $DB(\mathcal{A})$ satisfies $cln(\mathcal{T})$. In particular, by Lemma 4.11, it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore a model of $\mathcal{T}$. However, by construction of $\mathcal{A}$, $(\mathsf{funct}\ Q)$ is not satisfied in $DB(\mathcal{A})$, and hence also not in $can(\mathcal{O})$, which means that $can(\mathcal{O})$ is not a model of $(\mathsf{funct}\ Q)$. □

Notice that the role inclusion assertion we are using in Theorem 4.21 is of the form $\mathcal{T} \models Q \sqsubseteq \neg Q$, and thus expresses the fact that role Q has an empty extension in every model of $\mathcal{T}$. Also, by Theorem 4.20, logical implication of role inclusion assertions can in turn be reduced to ontology satisfiability.

Hence, with the above results in place, in the following we can concentrate on ontology satisfiability only.

4.5 Computational Complexity

From the results in the previous subsections we can establish the computational complexity characterization for the classical DL reasoning problems for *DL-Lite*$_{\mathcal{A}}$.

Theorem 4.22. *In DL-Lite*$_{\mathcal{A}}$*, ontology satisfiability is in* AC^0 *in the size of the ABox (data complexity) and in* PTIME *in the size of the whole ontology (combined complexity).*

Proof. First, AC^0 data complexity follows directly from FOL-rewritability, since evaluating FOL queries/formulas over a model is in AC^0 in the size of the model [90,1]. As for the combined complexity, we have that $cln(\mathcal{T})$ is polynomially related to the size of the TBox $\mathcal{T}$ and hence $q_{unsat(\mathcal{T})}$ defined in algorithm Satisfiable is formed by a number of disjuncts that is polynomial in $\mathcal{T}$. Each disjunct can be evaluated separately and contains either 2 or 3 variables. Now, each disjunct can be evaluated by checking the formula under each of the n^3 possible assignments, where n is the size of the domain of $DB(\mathcal{A})$ [90]. Finally, once an assignment is fixed, the evaluation of the formula can be done in AC^0. As a result, we get the PTIME bound. □

Taking into account the reductions in Theorems 4.16, 4.17, 4.18, 4.19, 4.20, and 4.21, as a consequence of Theorem 4.22, we get the following result.

Theorem 4.23. *In DL-Lite*$_{\mathcal{A}}$*, (concept and role) satisfiability and subsumption and logical implication of functionality assertions are in* PTIME *in the size of the TBox, and (concept and role) instance checking is in* AC^0 *in the size of the ABox and in* PTIME *in the size of the whole ontology.*

5 Query Answering over Ontologies

We study now query answering in *DL-Lite*$_{\mathcal{A},id}$. In a nutshell, our query answering method strongly separates the intensional and the extensional level of the *DL-Lite*$_{\mathcal{A},id}$ ontology: the query is first processed and reformulated based on the TBox axioms; then, the TBox is discarded and the reformulated query is evaluated over the ABox, as if the ABox was a simple relational database (cf. Section 2.6). More precisely, given a query q over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compile the assertions of $\mathcal{T}$ (in fact, the PIs in $\mathcal{T}$) into the query itself, thus obtaining a new query q'. Such a new query q' is then evaluated over $DB(\mathcal{A})$, thus essentially reducing query answering to query evaluation over a database instance. Since the size of q' does not depend on the ABox, the data complexity of the whole query answering algorithm is the same as the data complexity of evaluating q'. We show that, in the case where q is a CQ or a UCQ, the query q' is a UCQ. Hence, the data complexity of the whole query answering algorithm is AC^0.

As done in the previous section for ontology reasoning, we deal first with query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies. To this end, we establish some preliminary properties of *DL-Lite*$_{\mathcal{A}}$. Then we define an algorithm for the reformulation of CQs. Based on this algorithm we describe a technique for answering UCQs in *DL-Lite*$_{\mathcal{A}}$, prove its correctness, and analyze its computational complexity. Finally, we discuss the addition of identification assertions, and discuss query answering over *DL-Lite*$_{\mathcal{A},id}$ ontologies.

5.1 Preliminary Properties

First, we recall that, in the case where $\mathcal{O}$ is an unsatisfiable ontology, the answer to a UCQ q is the finite set of tuples *AllTup*$(q, \mathcal{O})$. Therefore, we focus for the moment on the case where $\mathcal{O}$ is satisfiable.

We start by showing a central property of the canonical interpretation $can(\mathcal{O})$. In particular, the following lemma shows that, for every model $\mathcal{M}$ of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, there is a homomorphism (cf. Definition 2.4) from $can(\mathcal{O})$ to $\mathcal{M}$.

Lemma 5.1. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology, and let $\mathcal{M} = (\Delta^{\mathcal{M}}, \cdot^{\mathcal{M}})$ be a model of $\mathcal{O}$. Then, there is a homomorphism from $can(\mathcal{O})$ to $\mathcal{M}$.*

Proof. We define a function ψ from $\Delta^{can(\mathcal{O})}$ to $\Delta^{\mathcal{M}}$ by induction on the construction of $chase(\mathcal{O})$, and simultaneously show that the following properties hold:

(i) for each atomic concept A in $\mathcal{O}$ and each object $o \in \Delta^{can(\mathcal{O})}$, if $o \in A^{can(\mathcal{O})}$ then $\psi(o) \in A^{\mathcal{M}}$, and

(ii) for each atomic role P in $\mathcal{O}$ and each pair of objects $o, o' \in \Delta^{can(\mathcal{O})}$, if $(o, o') \in P^{can(\mathcal{O})}$ then $(\psi(o), \psi(o')) \in P^{\mathcal{M}}$.

Hence, ψ is the desired homomorphism.

Base Step. For each constant d occurring in $\mathcal{A}$, we set $\psi(d^{can(\mathcal{O})}) = d^{\mathcal{M}}$ (notice that each model $\mathcal{M}$ interprets each such constant with an element in $\Delta^{\mathcal{M}}$). We remind that $chase_0(\mathcal{O}) = \mathcal{A}$, $\Delta^{can_0(\mathcal{O})} = \Delta^{can(\mathcal{O})} = \Gamma_C$, and that, for each constant d occurring in $\mathcal{A}$, $d^{can_0(\mathcal{O})} = d$. Then, it is easy to see that for each object $o \in \Delta^{can_0(\mathcal{O})}$ (resp., each pair of objects $o_1, o_2 \in \Delta^{can_0(\mathcal{O})}$) such that $o \in A^{can_0(\mathcal{O})}$, where A is an atomic concept in $\mathcal{O}$ (resp., $(o_1, o_2) \in P^{can(\mathcal{O})}$, where P is an atomic role in $\mathcal{O}$), we have that $A(o) \in chase_0(\mathcal{O})$ (resp., $P(o_1, o_2) \in chase_0(\mathcal{O})$). Since $\mathcal{M}$ satisfies all membership assertions in $\mathcal{A}$, we also have that $\psi(o) \in A^{\mathcal{M}}$ (resp., $(\psi(o_1), \psi(o_2)) \in P^{\mathcal{M}}$).

Inductive Step. Let us assume that $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by applying rule **cr2**. This means that a PI of the form $A \sqsubseteq \exists Q$, where A is an atomic concept in $\mathcal{T}$, and Q is a basic role in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exist a constant $a'' \in \Gamma_C$ such that $Q(a, a'') \in chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{Q(a, a')\}$, where a' follows lexicographically all constants appearing in $chase_i(\mathcal{O})$ (notice that this means that $(a, a') \in Q^{can_{i+1}(\mathcal{O})}$). By induction hypothesis, there exists $o_m \in \Delta^{\mathcal{M}}$ such that $\psi(a) = o_m$ and $o_m \in A^{\mathcal{M}}$. Because of the presence of the PI $A \sqsubseteq \exists Q$ in $\mathcal{T}$, and because $\mathcal{M}$ is a model of $\mathcal{O}$, there is at least one object $o'_m \in \Delta^{\mathcal{M}}$ such that $(o_m, o'_m) \in Q^{\mathcal{M}}$. Then, we set $\psi(a') = o'_m$, and we can conclude that $(\psi(a), \psi(a')) \in Q^{\mathcal{M}}$.

With a similar argument we can prove the inductive step also in those cases in which $can_{i+1}(\mathcal{O})$ is obtained from $can_i(\mathcal{O})$ by applying one of the rules **cr1**, **cr3**, **cr4**, or **cr5**. □

Based on the above property, we now prove that the canonical interpretation $can(\mathcal{O})$ of a satisfiable ontology $\mathcal{O}$ is able to represent all models of $\mathcal{O}$ with respect to UCQs.

Theorem 5.2. *Let $\mathcal{O}$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology, and let q be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = q^{can(\mathcal{O})}$.*

Proof. We first recall that $\Delta^{can(\mathcal{O})} = \Gamma_C$ and that, for each constant a occurring in $\mathcal{O}$, we have that $a^{can(\mathcal{O})} = a$. Therefore, given a tuple $\boldsymbol{t}$ of constants occurring in $\mathcal{O}$, we have that $\boldsymbol{t}^{can(\mathcal{O})} = \boldsymbol{t}$. We can hence rephrase the claim as $\boldsymbol{t} \in cert(q, \mathcal{O})$ iff $\boldsymbol{t} \in q^{can(\mathcal{O})}$.

"$\Rightarrow$" Suppose that $\boldsymbol{t} \in cert(q, \mathcal{O})$. Then, since $can(\mathcal{O})$ is a model of $\mathcal{O}$, we have that $\boldsymbol{t}^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$.

"$\Leftarrow$" Suppose that $\boldsymbol{t}^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$. Let q be the UCQ $q = \{q_1, \ldots, q_k\}$ with q_i defined as $q_i(\boldsymbol{x}_i) \leftarrow conj_i(\boldsymbol{x}_i, \boldsymbol{y}_i)$, for each $i \in \{1, \ldots, k\}$. Then, by Theorem 2.6, there exists $i \in \{1, \ldots, k\}$ such that there is a homomorphism μ from $conj_i(\boldsymbol{t}, \boldsymbol{y}_i)$ to $can(\mathcal{O})$ (cf. Definition 2.5).

Now let $\mathcal{M}$ be a model of $\mathcal{O}$. By Lemma 5.1, there is a homomorphism ψ from $can(\mathcal{O})$ to $\mathcal{M}$. Since homomorphisms are closed under composition, the function obtained by composing μ and ψ is a homomorphisms from $conj_i(\boldsymbol{t}, \boldsymbol{y}_i)$ to $\mathcal{M}$, and by Theorem 2.6, we have that $\boldsymbol{t}^{\mathcal{M}} \in q^{\mathcal{M}}$. Since $\mathcal{M}$ was an arbitrary model, this implies that $\boldsymbol{t} \in cert(q, \mathcal{O})$. □

The above property shows that the canonical interpretation $can(\mathcal{O})$ is a correct representative of all the models of a *DL-Lite*$_{\mathcal{A}}$ ontology with respect to the problem of answering UCQs. In other words, for every UCQ q, the answers to q over $\mathcal{O}$ correspond to the evaluation of q in $can(\mathcal{O})$.

In fact, this property holds for all positive FOL queries, but not in general. Consider for example the *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle\emptyset, \{A_1(a)\}\rangle$, and the FOL boolean query $q = \exists x. A_1(x) \wedge \neg A_2(x)$. We have that $chase(\mathcal{O}) = \{A_1(a)\}$, and therefore q is $true$ in $can(\mathcal{O})$, but the answer to q over $\mathcal{O}$ is $false$, since there exists a model $\mathcal{M}$ of $\mathcal{O}$ such that q is $false$ in $\mathcal{M}$. Assume, for instance, that $\mathcal{M}$ has the same interpretation domain as $can(\mathcal{O})$, and that $a^{\mathcal{M}} = a$, $A_1^{\mathcal{M}} = \{a\}$, and $A_2^{\mathcal{M}} = \{a\}$. It is easy to see that $\mathcal{M}$ is a model of $\mathcal{O}$ and that q is $false$ in $\mathcal{M}$.

Theorem 5.2, together with the fact that the canonical interpretation depends only on the positive inclusions, and not on the negative inclusions or the functionality assertions, has an interesting consequence for *satisfiable* ontologies, namely that the certain answers to a UCQ depend only on the set of positive inclusions and the ABox, but are not affected by the negative inclusions and the functionality assertions.

Corollary 5.3. *Let $\mathcal{O} = \langle\mathcal{T}, \mathcal{A}\rangle$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology, and let q be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = cert(q, \langle\mathcal{T}_p, \mathcal{A}\rangle)$, where $\mathcal{T}_p$ is the set of positive inclusions in $\mathcal{T}$.*

We point out that the canonical interpretation is in general infinite, consequently it cannot be effectively computed in order to solve the query answering problem in *DL-Lite*$_{\mathcal{A}}$.

Now, given the limited expressive power of *DL-Lite*$_{\mathcal{A}}$ TBoxes, it might seem that, in order to answer a query over an ontology $\mathcal{O}$, we could simply build a *finite* interpretation $\mathcal{I}_{\mathcal{O}}$ that allows for reducing answering *every* UCQ (or even every single CQ) over $\mathcal{O}$ to evaluating the query in $\mathcal{I}_{\mathcal{O}}$. The following theorem shows that this is *not* the case.

Theorem 5.4. *There exists a DL-Lite$_{\mathcal{A}}$ ontology $\mathcal{O}$ for which no finite interpretation $\mathcal{I}_{\mathcal{O}}$ exists such that, for every CQ q over $\mathcal{O}$, $cert(q, \mathcal{O}) = q^{\mathcal{I}_{\mathcal{O}}}$.*

Proof. Let $\mathcal{O}$ be the *DL-Lite*$_{\mathcal{A}}$ ontology whose TBox consists of the cyclic concept inclusion $\exists P^- \sqsubseteq \exists P$ and whose ABox consists of the assertion $P(a,b)$.

Let $\mathcal{I}_{\mathcal{O}}$ be a finite interpretation. There are two possible cases:

(1) There is no cycle on the relation P in $\mathcal{I}_{\mathcal{O}}$, i.e., the maximum path on the relation $P^{\mathcal{I}_{\mathcal{O}}}$ has a finite length n. In this case, consider the boolean conjunctive query $q() \leftarrow P(x_0,x_1), P(x_1,x_2), \ldots, P(x_n,x_{n+1})$ that represents the existence of a path of length $n+1$ in P. It is immediate to verify that the query q is *false* in $\mathcal{I}_{\mathcal{O}}$, i.e., $q^{\mathcal{I}_{\mathcal{O}}} = \emptyset$, while the answer to q over $\mathcal{O}$ is *true*, i.e., $cert(q,\mathcal{O}) = () \neq \emptyset$. This last property can be seen easily by noticing that $q^{can(\mathcal{O})}$ is *true*.

(2) $\mathcal{I}_{\mathcal{O}}$ satisfies the TBox cycle, so it has a finite cycle. More precisely, let us assume that $\mathcal{I}_{\mathcal{O}}$ is such that $\{(o_1,o_2),(o_2,o_3),\ldots,(o_n,o_1)\} \subseteq P^{\mathcal{I}_{\mathcal{O}}}$. In this case, consider the boolean CQ $q() \leftarrow P(x_1,x_2),\ldots,P(x_n,x_1)$. It is immediate to verify that such a query is *true* in $\mathcal{I}_{\mathcal{O}}$, while the answer to q over $\mathcal{O}$ is *false*. This last property can be seen easily by noticing that $q^{can(\mathcal{O})}$ is *false*, since $chase(\mathcal{O})$ does not contain a set of facts $P(a_1,a_2), P(a_2,a_3),\ldots,P(a_n,a_1)$, for any n, and therefore in $can(\mathcal{O})$ there does not exist any cycle on the relation P.

Consequently, in both cases $cert(q,O) \neq q^{\mathcal{I}_{\mathcal{O}}}$. □

Notice that in the proof of the above result we are using neither functionality assertions nor role inclusion assertions, hence the results holds already for the fragment of *DL-Lite*$_{\mathcal{A}}$ called *DL-Lite*$_{core}$ [24]. The above property demonstrates that answering queries in *DL-Lite*$_{\mathcal{A}}$ (or even in *DL-Lite*$_{core}$) goes beyond both propositional logic and relational databases.

Finally, we prove a property that relates answering UCQs to answering CQs.

Theorem 5.5. *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A}}$ ontology, and let q be a UCQ over $\mathcal{O}$. Then,*

$$cert(q,\mathcal{O}) = \bigcup_{q_i \in q} cert(q_i,\mathcal{O}).$$

Proof. The proof that $\bigcup_{q_i \in q} cert(q_i,\mathcal{O}) \subseteq cert(q,\mathcal{O})$ is immediate. To prove that $cert(q,\mathcal{O}) \subseteq \bigcup_{q_i \in Q} cert(q_i,\mathcal{O})$, we distinguish two possible cases:

(1) $\mathcal{O}$ is unsatisfiable. Then, it immediately follows that $\bigcup_{q_i \in q} cert(q_i,\mathcal{O})$ and $cert(q,\mathcal{O})$ are equal and coincide with the set *AllTup*$(q,\mathcal{O})$;

(2) $\mathcal{O}$ is satisfiable. Suppose that every $q_i \in q$ is of the form $q_i(\boldsymbol{x}) \leftarrow conj_i(\boldsymbol{x},\boldsymbol{y}_i)$, and consider a tuple $\boldsymbol{t} \in cert(q,\mathcal{O})$. Then, by Theorem 5.2, $\boldsymbol{t}^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$, which implies that there exists $i \in \{1,\ldots,k\}$ such that $\boldsymbol{t}^{can(\mathcal{O})} \in conj_i(\boldsymbol{t},\boldsymbol{y}_i)^{can(\mathcal{O})}$. Hence, from Theorem 5.2, it follows that $\boldsymbol{t} \in cert(q_i,\mathcal{O})$. □

Informally, the above property states that the set of answers to a UCQ q in *DL-Lite*$_{\mathcal{A}}$ corresponds to the union of the answers to the various CQs in q.

5.2 Query Reformulation

Based on the properties shown above, we define now an algorithm for answering UCQs in *DL-Lite*$_{\mathcal{A}}$, and analyze then its computational complexity. We need some preliminary definitions.

Atom g	Positive inclusion α	$gr(g,\alpha)$
$A(x)$	$A_1 \sqsubseteq A$	$A_1(x)$
$A(x)$	$\exists P \sqsubseteq A$	$P(x, \_)$
$A(x)$	$\exists P^- \sqsubseteq A$	$P(\_, x)$
$P(x, \_)$	$A \sqsubseteq \exists P$	$A(x)$
$P(x, \_)$	$\exists P_1 \sqsubseteq \exists P$	$P_1(x, \_)$
$P(x, \_)$	$\exists P_1^- \sqsubseteq \exists P$	$P_1(\_, x)$
$P(\_, x)$	$A \sqsubseteq \exists P^-$	$A(x)$
$P(\_, x)$	$\exists P_1 \sqsubseteq \exists P^-$	$P_1(x, \_)$
$P(\_, x)$	$\exists P_1^- \sqsubseteq \exists P^-$	$P_1(\_, x)$
$P(x_1, x_2)$	$P_1 \sqsubseteq P$ or $P_1^- \sqsubseteq P^-$	$P_1(x_1, x_2)$
$P(x_1, x_2)$	$P_1 \sqsubseteq P^-$ or $P_1^- \sqsubseteq P$	$P_1(x_2, x_1)$

Fig. 12. The result $gr(g,\alpha)$ of applying a positive inclusion α to an atom g

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant. Instead, an argument of an atom in a query is *unbound* if it corresponds to a non-distinguished non-shared variable. As usual, we use the symbol '$\_$' to represent non-distinguished non-shared variables.

We define first when a PI is *applicable to an atom*:

- A PI α is applicable to an atom $A(x)$, if α has A in its right-hand side.
- A PI α is applicable to an atom $P(x_1, x_2)$, if one of the following conditions holds:
  (i) $x_2 = \_$ and the right-hand side of α is $\exists P$; or
  (ii) $x_1 = \_$ and the right-hand side of α is $\exists P^-$; or
  (iii) α is a role inclusion assertion and its right-hand side is either P or P^-.

Roughly speaking, a PI α is applicable to an atom g if the predicate of g is equal to the predicate in the right-hand side of α and, in the case when α is an inclusion assertion between concepts, if g has at most one bound argument that corresponds to the object that is implicitly referred to by the inclusion α.

We indicate with $gr(g,\alpha)$ the atom obtained from the atom g by applying the applicable inclusion α. Formally:

Definition 5.6. *Let α be an PI that is applicable to the atom g. Then, $gr(g,\alpha)$ is the atom obtained from g and α as defined in Figure 12.*

In Figure 13, we provide the algorithm PerfectRef, which reformulates a UCQ (considered as a set of CQs) by taking into account the PIs of a TBox $\mathcal{T}$. In the algorithm, $q'[g/g']$ denotes the CQ obtained from a CQ q' by replacing the atom g with a new atom g'. Furthermore, *anon* is a function that takes as input a CQ q' and returns a new CQ obtained by replacing each occurrence of an unbound variable in q' with the symbol $\_$. Finally, *reduce* is a function that takes as input a CQ q' and two atoms g_1 and g_2 occurring in the body of q', and returns a CQ q'' obtained by applying to q' the *most general unifier* between g_1 and g_2. We point out that, in unifying g_1 and g_2, each occurrence of the $\_$ symbol has to be considered a different unbound variable. The most

```
Algorithm PerfectRef$(q, \mathcal{T})$
Input: UCQ $q$, $DL\text{-}Lite_{\mathcal{A}}$ TBox $\mathcal{T}$
Output: UCQ $pr$
$pr := q$;
repeat
  $pr' := pr$;
  for each CQ $q' \in pr'$ do
  (a) for each atom $g$ in $q'$ do
        for each PI $\alpha$ in $\mathcal{T}$ do
          if $\alpha$ is applicable to $g$
          then $pr := pr \cup \{ q'[g/gr(g, \alpha)] \}$;
  (b) for each pair of atoms $g_1, g_2$ in $q'$ do
        if $g_1$ and $g_2$ unify
        then $pr := pr \cup \{anon(reduce(q', g_1, g_2))\}$;
until $pr' = pr$;
return $pr$;
```

Fig. 13. The algorithm PerfectRef that computes the perfect reformulation of a CQ w.r.t. a *DL-Lite*$_{\mathcal{A}}$ TBox

general unifier substitutes each _ symbol in g_1 with the corresponding argument in g_2, and vice-versa (obviously, if both arguments are _, the resulting argument is _).

Informally, the algorithm first reformulates the atoms of each CQ $q' \in pr'$, and produces a new query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, which allow one to compile away in the reformulation the intensional knowledge (represented by $\mathcal{T}$) that is relevant for answering q. At step (b), for each pair of atoms g_1, g_2 that unify and occur in the body of a query q', the algorithm computes the CQ $q'' = reduce(q, g_1, g_2)$. Thanks to the unification performed by *reduce*, variables that are bound in q' may become unbound in q''. Hence, PIs that were not applicable to atoms of q', may become applicable to atoms of q'' (in the next executions of step (a)). Notice that the use of *anon* is necessary in order to guarantee that each unbound variable is represented by the symbol _.

We observe that the reformulation of a UCQ q w.r.t. a TBox $\mathcal{T}$ computed by PerfectRef depends only on the set of PIs in $\mathcal{T}$, and that NIs and functionality assertions do not play any role in such a process. Indeed, as demonstrated below by the proof of correctness of answering (U)CQs over *DL-Lite*$_{\mathcal{A}}$ ontologies based on the perfect reformulation, NIs and functionality assertions have to be considered only when verifying the satisfiability of the ontology. Once the satisfiability is established, they can be ignored in the query reformulation phase.

Example 5.7. Consider the following *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}_u$

$$\begin{array}{rlcrl}
\mathit{Professor} & \sqsubseteq \neg \mathit{Student} & \quad & \exists \mathit{HAS\text{-}TUTOR}^- & \sqsubseteq \mathit{Professor} \\
\mathit{Professor} & \sqsubseteq \exists \mathit{TEACHES\text{-}TO} & & \exists \mathit{TEACHES\text{-}TO}^- & \sqsubseteq \mathit{Student} \\
\mathit{Student} & \sqsubseteq \exists \mathit{HAS\text{-}TUTOR} & & & (\mathsf{funct}\ \mathit{HAS\text{-}TUTOR})
\end{array}$$

making use of the atomic concepts *Professor* and *Student*, and of the atomic roles *TEACHES-TO* and *HAS-TUTOR*. Such a TBox states that no student is also a professor (and vice-versa), that professors do teach to students, that students have a tutor, who is also a professor, and that everyone has at most one tutor.

Consider the CQ over $\mathcal{T}_u$

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{TEACHES\text{-}TO}(\_, y).$$

In such a query, the atoms $\mathit{TEACHES\text{-}TO}(x, y)$ and $\mathit{TEACHES\text{-}TO}(\_, y)$ unify, and by executing $\mathit{reduce}(q, \mathit{TEACHES\text{-}TO}(x, y), \mathit{TEACHES\text{-}TO}(\_, y))$, we obtain the atom $\mathit{TEACHES\text{-}TO}(x, y)$. The variable y is unbound, and therefore the function *anon* replaces it with $\_$. Now, the PI $\mathit{Professor} \sqsubseteq \exists \mathit{TEACHES\text{-}TO}$ can be applied to $\mathit{TEACHES\text{-}TO}(x, \_)$, whereas, before the reduction step, it could not be applied to any atom of the query. ■

The following lemma shows that the algorithm PerfectRef terminates, when applied to a UCQ and a $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ TBox.

Lemma 5.8. *Let $\mathcal{T}$ be a $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ TBox and q a UCQ over $\mathcal{T}$. Then, the algorithm* PerfectRef$(q, \mathcal{T})$ *terminates.*

Proof. The termination of PerfectRef, for each q and $\mathcal{T}$ given as inputs, immediately follows from the following facts:

(1) The maximum number of atoms in the body of a CQ generated by the algorithm is equal to the maximum length of the CQs in the input UCQ q. Indeed, in each iteration, a query atom is either replaced with another one, or the number of atoms in the query is reduced; hence, the number of atoms is bounded by the number of atoms in each input CQ. The length of the query is less than or equal to n, where n is the input query size, i.e., n is proportional to the number of atoms and the number of terms occurring in the input UCQ.
(2) The set of terms that occur in the CQs generated by the algorithm is equal to the set of variables and constants occurring in q plus the symbol $\_$, hence such a set has cardinality less than or equal to $n + 1$, where n is the query size.
(3) As a consequence of the above point, the number of different atoms that may occur in a CQ generated by the algorithm is less than or equal to $m \cdot (n + 1)^2$, where m is the number of predicate symbols (concept or role names) that occur either in the TBox or in the query.
(4) The algorithm does not drop queries that it has generated.

The above points 1 and 3 imply that the number of distinct CQs generated by the algorithm is finite, whereas point 4 implies that the algorithm does not generate a query more than once, and therefore PerfectRef terminates. More precisely, the number of distinct CQs generated by the algorithm is less than or equal to $(m \cdot (n + 1)^2)^n$, which corresponds to the maximum number of executions of the repeat-until cycle of the algorithm. □

Example 5.9. Consider the TBox $\mathcal{T}_u$ in Example 5.7 and the CQ

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{HAS\text{-}TUTOR}(y, \_)$$

asking for professors that teach to students that have a tutor.

Let us analyze the execution of the algorithm $\mathsf{PerfectRef}(\{q\}, \mathcal{T}_u)$. At the first execution of step (a), the algorithm applies to the atom $\mathit{HAS\text{-}TUTOR}(y, \_)$ the PI $\mathit{Student} \sqsubseteq \exists \mathit{HAS\text{-}TUTOR}$ and inserts in pr the new query

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{Student}(y).$$

Then, at a second execution of step (a), the algorithm applies to the atom $\mathit{Student}(y)$ the PI $\exists \mathit{TEACHES\text{-}TO}^- \sqsubseteq \mathit{Student}$ and inserts in pr the query

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{TEACHES\text{-}TO}(\_, y).$$

Since the two atoms of the second query unify, step (b) of the algorithm inserts into pr the query

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, \_).$$

Notice that the variable y is unbound in the new query, hence it has been replaced by the symbol $\_$. At a next iteration, step (a) applies $\mathit{Professor} \sqsubseteq \exists \mathit{TEACHES\text{-}TO}$ to $\mathit{TEACHES\text{-}TO}(x, \_)$ and inserts into pr the query

$$q(x) \leftarrow \mathit{Professor}(x).$$

Then, at a further execution of step (a), it applies $\exists \mathit{HAS\text{-}TUTOR}^- \sqsubseteq \mathit{Professor}$ to $\mathit{Professor}(x)$ and inserts into pr the query

$$q(x) \leftarrow \mathit{HAS\text{-}TUTOR}(\_, x).$$

The set constituted by the above five queries and the original query q is then returned by the algorithm $\mathsf{PerfectRef}(\{q\}, \mathcal{T}_u)$. ■

Example 5.10. As a further example, consider now the TBox $\mathcal{T}'_u$ obtained from $\mathcal{T}_u$ in Example 5.7 by adding the role inclusion assertion $\mathit{HAS\text{-}TUTOR}^- \sqsubseteq \mathit{TEACHES\text{-}TO}$, expressing that a tutor also teaches the student s/he is tutoring. Notice that $\mathcal{T}'_u$ is a valid $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ TBox. Consider the CQ

$$q'(x) \leftarrow \mathit{Student}(x).$$

Then, the result of $\mathsf{PerfectRef}(\{q'\}, \mathcal{T}'_u)$ is the UCQ

$$\begin{aligned} q'(x) &\leftarrow \mathit{Student}(x) \\ q'(x) &\leftarrow \mathit{TEACHES\text{-}TO}(\_, x) \\ q'(x) &\leftarrow \mathit{HAS\text{-}TUTOR}(x, \_) \end{aligned}$$

Notice that the insertion of the last CQ in the result of the execution of the algorithm is due to an application of the role inclusion assertion. ■

We note that the UCQ produced by PerfectRef is not necessarily minimal, i.e., it may contain pairs of CQs that are one contained into the other. Though this does not affect the worst-case computational complexity, for practical purposes this set of queries can be simplified, using well-known minimization techniques for relational queries. For example, it is possible to check ordinary containment between each pair of CQs in the produced UCQs, and remove from the result UCQ those CQs that are contained in some other CQ in the set. It is easy to see that such an optimization step will not affect completeness of the algorithm.

5.3 Query Evaluation

In order to compute the certain answers to a UCQ q over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we need to evaluate the set pr of CQs produced by $\mathsf{PerfectRef}(q, \mathcal{O})$ over the ABox $\mathcal{A}$ considered as a relational database.

In Figure 14, we define the algorithm Answer that, given a ontology $\mathcal{O}$ and a UCQ q, computes $cert(q, \mathcal{O})$. The following theorem shows that the algorithm Answer terminates, when applied to a UCQ and a *DL-Lite*$_{\mathcal{A}}$ TBox.

Theorem 5.11. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A}}$ ontology and q a UCQ over $\mathcal{O}$. Then, the algorithm* $\mathsf{Answer}(q, \mathcal{O})$ *terminates.*

Proof. Termination of $\mathsf{Answer}(q, \mathcal{O})$ follows straightforwardly from Lemma 4.13 and Lemma 5.8, which respectively establish termination of the algorithms $\mathsf{Satisfiable}(\mathcal{O})$ and $\mathsf{PerfectRef}(q, \mathcal{T})$. □

Example 5.12. Let us consider again the query of Example 5.9

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{HAS\text{-}TUTOR}(y, \_)$$

expressed over the ontology $\mathcal{O}_u = \langle \mathcal{T}_u, \mathcal{A}_u \rangle$, where $\mathcal{T}_u$ is the TBox defined in Example 5.7, and $\mathcal{A}_u$ consists of the membership assertions

$$\mathit{Student}(\mathit{john}), \qquad \mathit{HAS\text{-}TUTOR}(\mathit{john}, \mathit{mary}), \qquad \mathit{TEACHES\text{-}TO}(\mathit{mary}, \mathit{bill}).$$

By executing $\mathsf{Answer}(\{q\}, \mathcal{O}_u)$, since $\mathcal{O}_u$ is satisfiable (see Section 4), it executes $\mathsf{PerfectRef}(\{q\}, \mathcal{T}_u)$, which returns the UCQ described in Example 5.9. Let q_1 be such a query, then it is easy to see that $q_1^{DB(\mathcal{A}_u)}$ is the set $\{\mathit{mary}\}$.

Let us now consider again the query

$$q'(x) \leftarrow \mathit{Student}(x)$$

expressed over the ontology $\mathcal{O}'_u = \langle \mathcal{T}'_u, \mathcal{A}'_u \rangle$, where $\mathcal{T}'_u$ is as in Example 5.10, and $\mathcal{A}'_u$ consists of the membership assertions

$$\mathit{HAS\text{-}TUTOR}(\mathit{john}, \mathit{mary}), \qquad \mathit{TEACHES\text{-}TO}(\mathit{mary}, \mathit{bill}).$$

Obviously, $\mathcal{O}'_u$ is satisfiable, and executing $\mathsf{Answer}(\{q'\}, \mathcal{O}'_u)$ results in the evaluation of the UCQs returned by $\mathsf{PerfectRef}(\{q'\}, \mathcal{T}'_u)$, and which we have described in Example 5.10, over $\mathcal{A}'_u$. This produces the answer set $\{\mathit{john}, \mathit{bill}\}$. Notice that, without considering the additional role inclusion assertion, we would have obtained only $\{\mathit{bill}\}$ as answer to the query. ■

```
Algorithm Answer$(q, \mathcal{O})$
Input: UCQ $q$, DL-Lite$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: $cert(q, \mathcal{O})$
if not Satisfiable$(\mathcal{O})$
then return $AllTup(q, \mathcal{O})$;
else return $(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})}$;
```

Fig. 14. The algorithm Answer that computes the certain answers to a UCQ over a *DL-Lite*$_{\mathcal{A}}$ ontology

5.4 Correctness

We now prove correctness of the query answering technique described above. As discussed, from Theorem 5.2 it follows that query answering can in principle be done by evaluating the query over the model $can(\mathcal{O})$. However, since $can(\mathcal{O})$ is in general infinite, we obviously need to avoid the construction of $can(\mathcal{O})$. Instead, we compile the TBox into the query, thus simulating the evaluation of the query over $can(\mathcal{O})$ by evaluating a finite reformulation of the query over the ABox considered as a database.

Lemma 5.13. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, q a UCQ over $\mathcal{T}$, and pr the UCQ returned by* PerfectRef$(q, \mathcal{T})$. *For every DL-Lite$_\mathcal{A}$ ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = pr^{DB(\mathcal{A})}$.*

Proof. We first introduce the notion of witness of a tuple of constants with respect to a CQ. For a CQ $q'(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$, we denote with $conj'(\boldsymbol{x}, \boldsymbol{y})$ the set of atoms corresponding to the conjunction $conj(\boldsymbol{x}, \boldsymbol{y})$. Given a *DL-Lite$_\mathcal{A}$* knowledge base $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, a CQ $q'(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$ over $\mathcal{O}$, and a tuple $\boldsymbol{t}$ of constants occurring in $\mathcal{O}$, a set of membership assertions $\mathcal{G}$ is a *witness of $\boldsymbol{t}$ w.r.t. q'* if there exists a substitution σ from the variables $\boldsymbol{y}$ in $conj'(\boldsymbol{t}, \boldsymbol{y})$ to constants in $\mathcal{G}$ such that the set of atoms in $\sigma(conj'(\boldsymbol{t}, \boldsymbol{y}))$ is equal to $\mathcal{G}$. In particular, we are interested in witnesses of a tuple $\boldsymbol{t}$ w.r.t. a CQ q' that are contained in $chase(\mathcal{O})$. Intuitively, each such witness corresponds to a subset of $chase(\mathcal{O})$ that is sufficient in order to have that the formula $\exists \boldsymbol{y}.\, conj(\boldsymbol{t}, \boldsymbol{y})$ evaluates to $true$ in the canonical interpretation $can(\mathcal{O})$, and therefore the tuple $\boldsymbol{t} = \boldsymbol{t}^{can(\mathcal{O})}$ belongs to $q'^{can(\mathcal{O})}$. More precisely, we have that $\boldsymbol{t} \in q'^{can(\mathcal{O})}$ iff there exists a witness $\mathcal{G}$ of $\boldsymbol{t}$ w.r.t. q' such that $\mathcal{G} \subseteq chase(\mathcal{O})$. The cardinality of a witness $\mathcal{G}$, denoted by $|\mathcal{G}|$, is the number of membership assertions in $\mathcal{G}$.

Since $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, by Theorem 5.2, $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = q^{can(\mathcal{O})}$, and, by Theorem 5.5, $pr^{DB(\mathcal{A})} = \bigcup_{\overline{q} \in pr} \overline{q}^{DB(\mathcal{A})}$, where pr is the UCQ returned by PerfectRef$(q, \mathcal{T})$. Consequently, to prove the claim it is sufficient to show that $\bigcup_{\overline{q} \in pr} \overline{q}^{DB(\mathcal{A})} = q^{can(\mathcal{O})}$. We show both inclusions separately.

"$\subseteq$" We have to prove that $\overline{q}^{DB(\mathcal{A})} \subseteq q^{can(\mathcal{O})}$, for each CQ $\overline{q} \in pr$. We show by induction on the number of steps (a) and (b) executed by the algorithm PerfectRef to obtain $\overline{q}$ that $\overline{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. The claim then follows from the fact that $DB(\mathcal{A})$ is contained in $can(\mathcal{O})$ and that CQs are monotone.

Base step: trivial, since $\overline{q} \in q$.

Inductive step: Let the CQ $\overline{q} = q_{i+1}$ be obtained from q_i by means of step (a) or step (b) of the algorithm PerfectRef. We show in both cases that $q_{i+1}^{can(\mathcal{O})} \subseteq q_i^{can(\mathcal{O})}$. By the inductive hypothesis we then have that $q_i^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$, and the claim follows. We first consider the case in which q_{i+1} is obtained from q_i by applying step (a) of the algorithm. Let $\boldsymbol{t}$ be a tuple of constants occurring in $\mathcal{O}$ such that $\boldsymbol{t}^{can(\mathcal{O})} \in q_{i+1}^{can(\mathcal{O})}$. Then, it follows that there exists $\mathcal{G} \subseteq can(\mathcal{O})$ such that $\mathcal{G}$ is a witness of $\boldsymbol{t}$ w.r.t. q_{i+1}. Let us assume that q_{i+1} is obtained from q_i by applying step (a) when the positive inclusion assertion α of $\mathcal{T}$ is of the form $A_1 \sqsubseteq A$, i.e., $q_{i+1} = q_i[A(x)/A_1(x)]$ (the proof when α is of the other forms listed in Definition 5.6 is analogous). Then, either $\mathcal{G}$ is a witness of $\boldsymbol{t}$ w.r.t. q_i, or there exists a membership assertion in $\mathcal{G}$ to which the PI $A_1 \sqsubseteq A$ is applicable. In both cases there exists a witness of $\boldsymbol{t}$ w.r.t. q_i contained in $chase(\mathcal{O})$.

Therefore, $\boldsymbol{t}^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. We consider now the case in which q_{i+1} is obtained from q_i by applying step (b) of the algorithm, i.e., $q_{i+1} = \mathit{anon}(\mathit{reduce}(q_i, g_1, g_2))$, where g_1 and g_2 are two atoms belonging to q_i that unify. It is easy to see that in such a case $\mathcal{G}$ is also a witness of $\boldsymbol{t}$ w.r.t. q_i, and therefore $\boldsymbol{t}^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$.

"$\supseteq$" We have to show that for each tuple $\boldsymbol{t} \in q^{can(\mathcal{O})}$, there exists $\overline{q} \in pr$ such that $\boldsymbol{t} \in \overline{q}^{DB(\mathcal{A})}$. First, since $\boldsymbol{t} \in q^{can(\mathcal{O})}$, it follows that there exists a CQ q_0 in q and a finite number k such that there is a witness $\mathcal{G}_k$ of $\boldsymbol{t}$ w.r.t. q_0 contained in $chase_k(\mathcal{O})$. Moreover, without loss of generality, we can assume that every rule **cr1**, **cr2**, **cr3**, **cr4**, and **cr5** used in the construction of $chase(\mathcal{O})$ is necessary in order to generate such a witness $\mathcal{G}_k$: i.e., $chase_k(\mathcal{O})$ can be seen as a forest (set of trees) where: *(i)* the roots correspond to the membership assertions of $\mathcal{A}$; *(ii)* $chase_k(\mathcal{O})$ contains exactly k edges, where each edge corresponds to an application of a rule; *(iii)* each leaf is either one of the roots or a membership assertion in $\mathcal{G}_k$. In the following, we say that a membership assertion β is an *ancestor* of a membership assertion β' in a set of membership assertions $\mathcal{S}$, if there exist $\beta_1, \ldots, \beta_n$ in $\mathcal{S}$, such that $\beta_1 = \beta$, $\beta_n = \beta'$, and each β_i can be generated by applying a chase rule to β_{i-1}, for $i \in \{2, \ldots, n\}$. We also say that β' is a successor of β. Furthermore, for each $i \in \{0, \ldots, k\}$, we denote with $\mathcal{G}_i$ the *pre-witness* of $\boldsymbol{t}$ w.r.t. q in $chase_i(\mathcal{O})$, defined as follows:

$$\mathcal{G}_i = \bigcup_{\beta' \in \mathcal{G}_k} \{\, \beta \in chase_i(\mathcal{O}) \mid \begin{array}{l} \beta \text{ is an ancestor of } \beta' \text{ in } chase_k(\mathcal{O}) \text{ and} \\ \text{there exists no successor of } \beta \text{ in } chase_i(\mathcal{O}) \\ \text{that is an ancestor of } \beta' \text{ in } chase_k(\mathcal{O}) \,\}. \end{array}$$

Now we prove by induction on i that, starting from $\mathcal{G}_k$, we can "go back" through the rule applications and find a query $\overline{q}$ in pr such that the pre-witness $\mathcal{G}_{k-i}$ of $\boldsymbol{t}$ w.r.t. q_0 in $chase_{k-i}(\mathcal{O})$ is also a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$. To this aim, we prove that there exists $\overline{q} \in pr$ such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_{k-i}|$, where $|\overline{q}|$ indicates the number of atoms in the CQ $\overline{q}$. The claim then follows for $i = k$, since $chase_0(\mathcal{O}) = \mathcal{A}$.

Base step: There exists $\overline{q} \in pr$ such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_k|$. This is an immediate consequence of the fact that $q_0 \in pr$ and that pr is closed with respect to step (b) of the algorithm PerfectRef. Indeed, if $|\mathcal{G}_k| < |q_0|$ then there exist two atoms g_1, g_2 in q_0 and a membership assertion β in $\mathcal{G}_k$ such that β and g_1 unify and β and g_2 unify, which implies that g_1 and g_2 unify. Therefore, by step (b) of the algorithm, it follows that there exists a query $q_1 \in pr$ (with $q_1 = \mathit{reduce}(q_0, g_1, g_2)$) such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. q_1 and $|q_1| = |q| - 1$. Now, if $|\mathcal{G}_k| < |q_1|$, we can iterate the above argument, thus we conclude that there exists $\overline{q} \in pr$ such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_k|$.

Inductive step: suppose that there exists $\overline{q} \in pr$ such that $\mathcal{G}_{k-i+1}$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_{k-i+1}|$. Let us assume that $chase_{k-i+1}(\mathcal{O})$ is obtained by applying **cr2** to $chase_{k-i}(\mathcal{O})$ (the proof is analogous for rules **cr1**, **cr3**, **cr4**, and **cr5**). This means that a PI of the form $A \sqsubseteq \exists P$[12], where A is an atomic concept and P is an atomic role, is applied in $chase_{k-i}(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exists $a' \in \Gamma_C$ such that $P(a, a') \in chase_{k-i}(\mathcal{O})$. Therefore,

[12] The other execution of rule **cr2** is for the case where the PI is $A \sqsubseteq \exists P^-$, which is analogous.

$chase_{k-i+1}(\mathcal{O}) = chase_{k-i}(\mathcal{O}) \cup \{P(a, a'')\}$, where $a'' \in \Gamma_C$ follows lexicographically all constants occurring in $chase_i(\mathcal{O})$.

Since a'' is a new constant of Γ_C, i.e., a constant not occurring elsewhere in $\mathcal{G}_{k-i+1}$, and since $|\overline{q}| = |\mathcal{G}_{k-i+1}|$, it follows that the atom $P(x, \_)$ occurs in $\overline{q}$. Therefore, by step (a) of the algorithm, it follows that there exists a query $q_1 \in pr$ (with $q_1 = \overline{q}[P(x, \_)/A(x)]$) such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. q_1.

Now, there are two possible cases: either $|q_1| = |\mathcal{G}_{k-i}|$, and in this case the claim is immediate; or $|q_1| = |\mathcal{G}_{k-i}| + 1$. This last case arises if and only if the membership assertion $A(a)$ to which the rule **cr2** is applied is both in $\mathcal{G}_{k-i}$ and in $\mathcal{G}_{k-i+1}$. This implies that there exist two atoms g_1 and g_2 in q_1 such that $A(a)$ and g_1 unify and $A(a)$ and g_2 unify, hence g_1 and g_2 unify. Therefore, by step (b) of the algorithm (applied to q_1), it follows that there exists $q_2 \in pr$ (with $q_2 = reduce(q_1, g_1, g_2)$) such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. q_2 and $|q_2| = |\mathcal{G}_{k-i+1}|$, which proves the claim. □

Based on the above property, we are finally able to establish correctness of the algorithm Answer.

Theorem 5.14. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology and q a UCQ. Then,* $cert(q, \mathcal{O}) = \mathsf{Answer}(q, \mathcal{O})$.

Proof. In the case where $\mathcal{O}$ is satisfiable, the proof follows immediately from Lemma 5.13 and Theorem 5.5. In the case where $\mathcal{O}$ is not satisfiable, it is immediate to verify that the set $AllTup(q, \mathcal{O})$ returned by $\mathsf{Answer}(q, \mathcal{O})$ corresponds to $cert(q, \mathcal{O})$, according to the semantics of queries given in Section 2.2. □

As an immediate corollary of the above properties, it follows that the problem of answering UCQs over satisfiable *DL-Lite*$_\mathcal{A}$ ontologies is FOL-rewritable. Moreover, it is easy to see that FOL-rewritability extends also to the case of arbitrary (both satisfiable and unsatisfiable) *DL-Lite*$_\mathcal{A}$ ontologies. Indeed, the whole query answering task can be encoded into a single UCQ, obtained by adding to the UCQ $\mathsf{PerfectRef}(q, \mathcal{T})$ a finite number of CQs encoding the fact that every tuple in $AllTup(q, \mathcal{O})$ is in the answer set of the query if $\mathcal{O}$ is unsatisfiable. (For details on the construction of such a query see e.g. [17], which defines a similar encoding in the context of relational database integrity constraints.) We therefore get the following theorem.

Theorem 5.15. *Answering UCQs in DL-Lite$_\mathcal{A}$ is FOL-rewritable.*

5.5 Computational Complexity

We first establish the complexity of the algorithm PerfectRef.

Lemma 5.16. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and q a UCQ over $\mathcal{T}$. The algorithm* $\mathsf{PerfectRef}(q, \mathcal{T})$ *runs in time polynomial in the size of $\mathcal{T}$.*

Proof. Let n be the query size, and let m be the number of predicate symbols (concept or role names) that occur either in the TBox or in the query. As shown in Lemma 5.8, the number of distinct CQs generated by the algorithm is less than or equal to $(m \cdot (n + 1)^2)^n$, which corresponds to the maximum number of executions of the repeat-until cycle of the algorithm. Since m is linearly bounded by the size of the TBox $\mathcal{T}$, while n does not depend on the size of $\mathcal{T}$, from the above argument it follows that the algorithm $\mathsf{PerfectRef}(q, \mathcal{T})$ runs in time polynomial in the size of $\mathcal{T}$. $\square$

Based on the above property, we are able to establish the complexity of answering UCQs in *DL-Lite*$_\mathcal{A}$.

Theorem 5.17. *Answering UCQs in DL-Lite*$_\mathcal{A}$ *is in* PTIME *in the size of the ontology, and in* AC^0 *in the size of the ABox (data complexity).*

Proof. The proof is an immediate consequence of the correctness of the algorithm Answer, established in Theorem 5.14, and the following facts: *(i)* Lemma 5.16, which implies that the query $\mathsf{PerfectRef}(q, \mathcal{T})$ can be computed in time polynomial in the size of the TBox and constant in the size of the ABox (data complexity). *(ii)* Theorem 4.22, which states the computational complexity of checking satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies. *(iii)* The fact that the evaluation of a UCQ over a database can be computed in AC^0 with respect to the size of the database (since UCQs are a subclass of FOL queries) [1]. $\square$

We are also able to characterize the combined complexity (i.e., the complexity w.r.t. the size of $\mathcal{O}$ and q) of answering UCQs in *DL-Lite*$_\mathcal{R}$.

Theorem 5.18. *Answering UCQs in DL-Lite*$_\mathcal{A}$ *is NP-complete in combined complexity.*

Proof. To prove membership in NP, observe that a version of the algorithm $\mathsf{PerfectRef}$ that nondeterministically returns only one of the CQs belonging to the reformulation of the input query, runs in nondeterministic polynomial time in combined complexity, since every query returned by $\mathsf{PerfectRef}$ can be generated after a polynomial number of transformations of one of the input CQs (i.e., after a polynomial number of executions of steps (a) and (b) of the algorithm). This allows the corresponding nondeterministic version of the algorithm Answer to run in nondeterministic polynomial time when the input is a boolean UCQ. NP-hardness follows directly from NP-hardness of CQ evaluation over relational databases [1]. $\square$

To summarize, the above results show a very nice computational behavior of queries in *DL-Lite*$_\mathcal{A}$: answering UCQs over ontologies expressed in such a logic is computationally no worse than standard UCQ answering (and containment) in relational databases.

5.6 Dealing with Identification Assertions

We address now the addition of identification assertions, and present a technique for satisfiability and query answering in *DL-Lite*$_{\mathcal{A},id}$. We start with the following result, which extends Lemma 4.11 holding for *DL-Lite*$_\mathcal{A}$ ontologies to ontologies that contain also identification assertions.

Lemma 5.19. *Let* $\mathcal{O}$ *be a DL-Lite*$_{\mathcal{A},id}$ *ontology. Then,* $can(\mathcal{O})$ *is a model of* $\mathcal{O}$ *if and only if* $\mathcal{O}$ *is satisfiable.*

Proof (sketch). "$\Rightarrow$" If $can(\mathcal{O})$ is a model of $\mathcal{O}$, then $\mathcal{O}$ is obviously satisfiable.

"$\Leftarrow$" Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable *DL-Lite*$_{\mathcal{A},id}$ ontology, and let us show that $can(\mathcal{O})$ satisfies all assertions in $\mathcal{O}$. By Lemma 4.11, it is sufficient to show that $can(\mathcal{O})$ satisfies all IdCs. Since roles occurring in IdCs cannot be specialized, it is easy to see that the following crucial property holds: for each basic role Q and for each constant a introduced in $chase(\mathcal{O})$ during a chase step, and hence present in $can(\mathcal{O})$, there is in $chase(\mathcal{O})$ at most one fact of the form $Q(a, a')$ and at most one fact of the form $Q(a', a)$, where a' is some constant originally present in $\mathcal{A}$ or introduced during a previous chase step. From this property, it immediately follows that an IdC[13] can *not* be violated by a constant introduced during the chase. On the other hand, if an IdC was violated in $chase(\mathcal{O})$, and hence in $can(\mathcal{O})$, by some pair of constants of $\mathcal{A}$, then such an IdC would be violated in every model of $\mathcal{A}$, and hence $\mathcal{O}$ would be unsatisfiable, thus contradicting the hypothesis. Consequently, no IdC of $\mathcal{O}$ is violated in $can(\mathcal{O})$, thus $can(\mathcal{O})$ is a model of all IdCs, and hence a model of $\mathcal{O}$. $\square$

The above lemma allows us to establish a fundamental "separation" property for IdCs, similar to the one for functionality assertions and negative inclusions stated in Theorem 4.12. However, instead to resorting to a notion of closure of a set of (identification) assertions, to check satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology we rely on the perfect reformulation of the query that expresses the violation of an identification assertion.

As a preliminary step, we associate to each IdC α a boolean CQ with an inequality $\delta(\alpha)$ that encodes the violation of α (similarly to what we have done for negative inclusions and functionality assertions). We make use of the following notation, where B is a basic concept and x a variable:

$$\gamma(B, x) = \begin{cases} A(x), & \text{if } B = A, \\ P(x, y_{new}), & \text{if } B = \exists P, \text{ where } y_{new} \text{ is a fresh variable,} \\ P(y_{new}, x), & \text{if } B = \exists P^-, \text{ where } y_{new} \text{ is a fresh variable.} \end{cases}$$

Then, given an IdC $\alpha = (\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$, we define the boolean CQ with inequality

$$\delta(\alpha) \ = \ \exists \boldsymbol{x}.\, \gamma(B, x) \land \gamma(B, x') \land x \neq x' \land \bigwedge_{1 \leq i \leq n} (\rho(\pi_i(x, x_i)) \land \rho(\pi_i(x', x_i))),$$

where $\boldsymbol{x}$ are all variables appearing in the atoms of $\delta(\alpha)$, and $\rho(\pi(x, y))$ is inductively defined on the structure of path π as follows:

(1) If $\pi = B_1? \circ \cdots \circ B_h? \circ Q \circ B'_1? \circ \cdots \circ B'_k?$ (with $h \geq 0$, $k \geq 0$), then

$$\rho(\pi(x, y)) = \gamma(B_1, x) \land \cdots \land \gamma(B_h, x) \land Q(x, y) \land \gamma(B'_1, y) \land \cdots \land \gamma(B'_k, y).$$

(2) If $\pi = \pi_1 \circ \pi_2$, where $length(\pi_1) = 1$ and $length(\pi_2) \geq 1$, then

$$\rho(\pi(x, y)) = \rho(\pi_1(x, z)) \land \rho(\pi_2(z, y)),$$

where z is a fresh variable symbol (i.e., a variable symbol not occurring elsewhere in the query).

[13] Recall that we consider only so-called local IdCs, which have at least one path of length 1.

```
**Algorithm** SatisfiableIdC($\mathcal{O}$)
**Input:** $DL\text{-}Lite_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$
**Output:** $true$ if $\mathcal{O}$ is satisfiable, $false$ otherwise
**begin**
  **if not** Satisfiable($\langle \mathcal{T}, \mathcal{A} \rangle$)
  **then return** $false$;
  **else begin**
    $q_{\mathcal{T}_{id}} := \{\bot\}$;
    **for each** $\alpha \in \mathcal{T}_{id}$ **do** $q_{\mathcal{T}_{id}} := q_{\mathcal{T}_{id}} \cup \{\delta(\alpha)\}$;
    $q_{unsat(\mathcal{T}_{id})} :=$ PerfectRefIdC($q_{\mathcal{T}_{id}}, \mathcal{T}$);
    **if** $q_{unsat(\mathcal{T}_{id})}^{DB(\mathcal{A})} = \emptyset$ **then return** $true$; **else return** $false$;
  **end**
**end**
```

Fig. 15. The algorithm SatisfiableIdC that checks satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology

Intuitively, $\delta(\alpha)$ encodes the violation of α by asking for the existence of two distinct instances of B identified, according to α, by the same set of objects.

Consider now a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$, where we have denoted the TBox of such an ontology as the union $\mathcal{T} \cup \mathcal{T}_{id}$ of a set $\mathcal{T}$ of *DL-Lite*$_{\mathcal{A}}$ inclusion and functionality assertions and of a set $\mathcal{T}_{id}$ of IdCs. Let us assume that $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable. In order to check the satisfiability of $\mathcal{O}$ (i.e., assess the impact on satisfiability of the IdCs), we can consider the *perfect reformulation* of the query $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\delta(\alpha)\}$ encoding the violation of all IdCs in $\mathcal{T}_{id}$. However, we need to consider a variation of the reformulation algorithm PerfectRef shown in Figure 13 that takes into account the presence of inequalities in $q_{\mathcal{T}_{id}}$. Such an algorithm, denoted PerfectRefIdC, considers the inequality predicate as a new primitive role, and never "reduces" variables occurring in inequality atoms, i.e., such variables are never transformed by unification steps into non-join variables (cf. Section 5.2). Exploiting Lemma 5.19, we can prove the following result.

Theorem 5.20. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology, let $\mathcal{T}_{id}$ be a set of IdCs, and let $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\delta(\alpha)\}$. Then the DL-Lite$_{\mathcal{A},id}$ ontology $\mathcal{O}_{id} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ is satisfiable if and only if* (PerfectRefIdC($q_{\mathcal{T}_{id}}$))$^{DB(\mathcal{A})} = \emptyset$.

We present in Figure 15 the algorithm SatisfiableIdC that checks the satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$. First, the algorithm uses the algorithm Satisfiable to check satisfiability of the ordinary *DL-Lite*$_{\mathcal{A}}$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ obtained from $\mathcal{O}$ by discarding all IdCs: if $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable, then also $\mathcal{O}$ is unsatisfiable. Otherwise, the algorithm first computes the union $q_{\mathcal{T}_{id}}$ of the queries $\delta(\alpha)$, for all IdCs $\alpha \in \mathcal{T}_{id}$, encoding the violation of all IdCs in $\mathcal{O}$. Then, it uses PerfectRefIdC to compute the query $q_{unsat(\mathcal{T}_{id})}$ corresponding to the *perfect reformulation* of $q_{\mathcal{T}_{id}}$ with respect to the TBox assertions in $\mathcal{T}$. Finally, the algorithm evaluates $q_{unsat(\mathcal{T}_{id})}$ over $DB(\mathcal{A})$, i.e., the ABox $\mathcal{A}$ considered as a relational database, (which can be done in AC^0 w.r.t. the size of $\mathcal{A}$) and checks whether such an evaluation returns the empty set (i.e., whether the boolean UCQ evaluates to *false*). If this is the case, then the algorithm returns $true$

(i.e., that $\mathcal{O}$ is satisfiable), since the ABox does not violate any IdC that is logically implied by $\mathcal{T} \cup \mathcal{T}_{id}$ (note that considering $\mathsf{PerfectRefIdC}(q, \mathcal{T})$ rather than simply q is essential for this). Instead, if the evaluation of $q_{unsat(\mathcal{T}_{id})}$ over $DB(\mathcal{A})$ returns $true$, then the ABox violates some IdC, and the algorithm reports that $\mathcal{O}$ is unsatisfiable by returning $false$.

The following lemma establishes the correctness of SatisfiableIdC.

Lemma 5.21. *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A},id}$ ontology. Then, the algorithm* $\mathsf{SatisfiableIdC}(\mathcal{O})$ *terminates, and $\mathcal{O}$ is satisfiable if and only if* $\mathsf{SatisfiableIdC}(\mathcal{O}) = true$.

Proof. Termination follows immediately from termination of PerfectRefIdC and of evaluation of a FOL query over a database. The correctness is an immediate consequence of Theorem 5.20.

Correctness of the algorithm, together with the fact that the perfect reformulation is independent of the ABox (see Section 5.2), and, according to Lemma 5.16 can be computed in PTIME in the size of the TBox, allows us to extend the complexity results of Theorem 4.22 for ontology satisfiability in *DL-Lite*$_{\mathcal{A}}$ also to *DL-Lite*$_{\mathcal{A},id}$ ontologies.

Theorem 5.22. *In DL-Lite$_{\mathcal{A},id}$, ontology satisfiability is FOL-rewritable, and hence in* AC^0 *in the size of the ABox (data complexity), and in* PTIME *in the size of the whole ontology (combined complexity).*

We observe that also for checking the satisfiability of a *DL-Lite*$_{\mathcal{A}}$ ontology, specifically with respect to negative inclusion assertions, we could have adopted an approach similar to the one presented in this subsection based on query reformulation, rather than the one presented in Section 4.2 based on computing the closure $cln(\mathcal{T})$ of the negative inclusions. Specifically, one can check that the query $q_{unsat(\mathcal{T})}$ computed by Algorithm Satisfiable in Figure 9 starting from $cln(\mathcal{T})$, actually corresponds to

$$\mathsf{PerfectRef}(\bigcup_{\alpha \in \mathcal{T}_n} \delta(\alpha), \mathcal{T}_p) \;\cup\; \bigcup_{\alpha \in \mathcal{T}_f} \delta(\alpha),$$

where $\mathcal{T}_p$, $\mathcal{T}_n$, and $\mathcal{T}_f$ are respectively the sets of positive inclusions, negative inclusions, and functionality assertions in $\mathcal{T}$.

We now turn our attention to query answering in the presence of identification assertions. To this aim, we observe that Lemma 5.1 and Theorem 5.2 hold also for *DL-Lite*$_{\mathcal{A},id}$ ontologies, from which we can derive the analogue of Corollary 5.3, establishing separability for query answering in *DL-Lite*$_{\mathcal{A},id}$.

Corollary 5.23. *Let $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A},id}$ ontology, and let q be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = cert(q, \langle \mathcal{T}_p, \mathcal{A} \rangle)$, where $\mathcal{T}_p$ is the set of positive inclusions in $\mathcal{T}$.*

Then, Lemma 5.13 does not depend on the presence of identification assertions, except for the fact that they may affect satisfiability of an ontology. Hence, for answering UCQs over a *DL-Lite*$_{\mathcal{A},id}$ ontology we can resort to the Algorithm AnswerIdC, shown in Figure 16, which is analogous to the Algorithm Answer shown in Figure 14, with

```
**Algorithm** $\mathsf{AnswerIdC}(q, \mathcal{O})$
**Input:** UCQ $q$, $\textit{DL-Lite}_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$
**Output:** $\mathit{cert}(q, \mathcal{O})$
**if not** $\mathsf{SatisfiableIdC}(\mathcal{O})$
**then return** $\mathit{AllTup}(q, \mathcal{O})$;
**else return** $(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})}$;
```

Fig. 16. The algorithm AnswerIdC that computes the certain answers to a UCQ over a *DL-Lite*$_{\mathcal{A},id}$ ontology

the only difference that now the satisfiability check is done by taking into account also identification assertions.

The following theorem establishes termination and correctness of the algorithm AnswerIdC, when applied to a UCQ and a *DL-Lite*$_{\mathcal{A},id}$ ontology.

Theorem 5.24. *Let* $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ *be a DL-Lite*$_{\mathcal{A},id}$ *ontology and q a UCQ. Then,* $\mathsf{AnswerIdC}(q, \mathcal{O})$ *terminates and* $\mathit{cert}(q, \mathcal{O}) = \mathsf{AnswerIdC}(q, \mathcal{O})$.

Also, we obtain for query answering over *DL-Lite*$_{\mathcal{A},id}$ ontologies exactly the same complexity bounds as for *DL-Lite*$_{\mathcal{A}}$ ontologies.

Theorem 5.25. *Answering UCQs in DL-Lite*$_{\mathcal{A},id}$ *is in* PTIME *in the size of the ontology, in* AC^0 *in the size of the ABox (data complexity), and NP-complete in combined complexity.*

Finally, it can be shown that adding identification assertions to *DL-Lite*$_{\mathcal{A}}$ does not increase the (data and combined) complexity of all other reasoning services, including logical implication of identification assertions.

6 Beyond *DL-Lite*$_{\mathcal{A},id}$

We now analyze the impact on the computational complexity of inference of extending the *DL-Lite*$_{\mathcal{A},id}$ DL as presented in Section 2. Specifically, we will concentrate on data complexity, and note that, whenever the data complexity of an inference problem goes beyond AC^0, then the problem is not FOL-rewritable. Hence, if we want to base inference on evaluating queries over a relational database, the lack of FOL-rewritability means that a more powerful query answering engine than those available in standard relational database technology is required. An immediate consequence of this fact is that we cannot take advantage anymore of data management tools and query optimization techniques of current DBMSs (cf. also Section 7).

There are two possible ways of extending *DL-Lite*$_{\mathcal{A},id}$. The first one corresponds to a proper language extension, i.e., adding new DL constructs to *DL-Lite*$_{\mathcal{A},id}$, while the second one consists of changing/strengthening the semantics of the formalism. We analyze both types of extensions.

6.1 Extending the Ontology Language

Concerning the extension of the *DL-Lite*$_{\mathcal{A},id}$ language, the results in [22], which we report below, and those in [4], show that, apart from number restrictions, it is not possible to add any of the usual DL constructs to *DL-Lite*$_{\mathcal{A},id}$ while keeping the data complexity of query answering within AC^0. This means that *DL-Lite*$_{\mathcal{A},id}$ is essentially the most expressive DL allowing for data integration systems where query answering is FOL-rewritable.

In addition to the constructs of *DL-Lite*$_{\mathcal{A},id}$, we consider here also the following common construct in DLs [7]:

– *concept conjunction*, denoted $C_1 \sqcap C_2$, and interpreted as $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$, for an interpretation $\mathcal{I}$;
– *concept disjunction*, denoted $C_1 \sqcup C_2$, and interpreted as $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$;
– *universal quantification on a role*, denoted $\forall P.A$, and interpreted as:

$$(\forall P.A)^{\mathcal{I}} \;=\; \{\, o \mid \forall o'.\, (o, o') \in P^{\mathcal{I}} \rightarrow o' \in A^{\mathcal{I}} \,\}.$$

We then consider variations of *DL-Lite*$_{\mathcal{A}}$ TBoxes, consisting of:

– concept inclusion assertions of the form $Cl \sqsubseteq Cr$, where the constructs that may occur in Cl and Cr will vary according to the language considered;
– possibly role inclusion assertions between atomic roles, i.e., of the form $P \sqsubseteq P'$;
– possibly functionality assertions of the form $(\mathsf{funct}\ P)$ and/or $(\mathsf{funct}\ P^-)$.

We first consider the case where we use qualified existential quantification in the left-hand side of inclusion assertions. This alone is sufficient to lose FOL-rewritability of instance checking. The same effect can be achieved with universal quantification on the right-hand side of inclusion assertions, or with functionality interacting with qualified existential on the right-hand side.

Theorem 6.1. *Instance checking (and hence ontology satisfiability and query answering) is* NLOGSPACE*-hard in data complexity for ontologies* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *where* $\mathcal{A}$ *is an ABox, and* $\mathcal{T}$ *is a TBox of one of the following forms:*

1. $Cl \longrightarrow A \mid \exists P.A$
   $Cr \longrightarrow A$
   Assertions in $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.
2. $Cl \longrightarrow A$
   $Cr \longrightarrow A \mid \forall P.A$
   Assertions in $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.
3. $Cl \longrightarrow A$
   $Cr \longrightarrow A \mid \exists P.A$
   Assertions in $\mathcal{T}$*:* $Cl \sqsubseteq Cr, \quad (\mathsf{funct}\ P)$.

Proof. For Case 1, the proof is by a LOGSPACE reduction of reachability in directed graphs, which is NLOGSPACE-complete [40], to instance checking. Let $G = \langle V, E \rangle$ be a directed graph, where V is a set of vertexes and $E \subseteq V \times V$ is a set of directed edges, and let s, t be two vertexes in V. *Reachability* is the problem of checking whether there are vertexes $v_0, v_1, \ldots, v_n$ in V with $v_0 = s$, $v_n = t$, and $(v_{i-1}, v_i) \in E$, for $i \in \{1, \ldots, n\}$, i.e., whether there is an oriented path formed by edges in E that, starting from s allows one to reach t.

We define an ontology $\mathcal{O} = \langle \mathcal{T}_{reach}, \mathcal{A}_G \rangle$, where the TBox $\mathcal{T}_{reach}$ is constituted by a single inclusion assertion

$$\exists P.A \sqsubseteq A$$

and the ABox $\mathcal{A}_G$ has as constants the nodes of G, and is constituted by the membership assertion $A(t)$, and by one membership assertion $P(v, v')$ for each edge $(v, v') \in E$. The TBox $\mathcal{T}_{reach}$ does not depend on G, and it is easy to see that $\mathcal{A}_G$ can be constructed in LOGSPACE from G, s, and t. We show that there is an oriented path in G from s to t if and only if $\mathcal{O} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in G from s to t. We construct a model $\mathcal{I}$ of $\mathcal{O}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = V$, $v^{\mathcal{I}} = v$ for each $v \in V$, $P^{\mathcal{I}} = E$, and $A^{\mathcal{I}} = \{ v \mid \text{ there is a path in } G \text{ from } v \text{ to } t \}$. We show that $\mathcal{I}$ is a model of $\mathcal{O}$. By construction, $\mathcal{I}$ satisfies all membership assertions $P(v, v')$ and the membership assertion $A(t)$. Consider an object $v \in (\exists P.A)^{\mathcal{I}}$. Then there is an object $v' \in A^{\mathcal{I}}$ such that $(v, v') \in P^{\mathcal{I}}$. Then, by definition of $\mathcal{I}$, there is a path in G from v' to t, and $(v, v') \in E$. Hence, there is also a path in G from v to t and, by definition of $\mathcal{I}$, we have that $v \in A^{\mathcal{I}}$. It follows that also the inclusion assertion $\exists P.A \sqsubseteq A$ is satisfied in $\mathcal{I}$.

"$\Rightarrow$" Suppose there is a path in G from a vertex v to t. We prove by induction on the length k of such a path that $\mathcal{O} \models A(v)$. Base case: $k = 0$, then $v = t$, and the claim follows from $A(t) \in \mathcal{A}_G$. Inductive case: suppose there is a path in G of length $k-1$ from v' to t and $(v, v') \in E$. By the inductive hypothesis, $\mathcal{O} \models A(v')$, and since by definition $P(v, v') \in \mathcal{A}$, we have that $\mathcal{O} \models \exists P.A(v)$. By the inclusion assertion in $\mathcal{T}_{reach}$ it follows that $\mathcal{O} \models A(v)$.

For Case 2, the proof follows from Case 1 and the observation that an assertion $\exists P.A_1 \sqsubseteq A_2$ is logically equivalent to the assertion $A_1 \sqsubseteq \forall P^-.A_2$, and that we can get rid of inverse roles by inverting the edges of the graph represented in the ABox.

For Case 3, the proof is again by a LOGSPACE reduction of reachability in directed graphs, and is based on the idea that an assertion $\exists P.A_1 \sqsubseteq A_2$ can be simulated by the assertions $A_1 \sqsubseteq \exists P^-.A_2$ and $(\mathsf{funct}\ P^-)$. Moreover, the graph can be encoded using only functional roles (see proof of Theorem 6.5), and we can again get rid of inverse roles by inverting edges. $\square$

Note that all the above "negative" results hold already for instance checking, i.e., for the simplest queries possible. Also, note that in all three cases, we are considering extensions to a minimal subset of *DL-Lite*$_{\mathcal{A},id}$ in order to get NLOGSPACE-hardness.

Notably, Case 3 of Theorem 6.1 tells us that instance checking (and therefore query answering), in the DL obtained from *DL-Lite*$_{\mathcal{A}}$ by removing the restriction on the interaction between functionality assertions and role inclusions (cf. Definition 2.1) is not in AC^0, and hence not FOL-rewritable. This can be seen easily by considering the encoding of inclusion assertions involving qualified existential restriction on the right-hand side in terms of inclusion assertions between roles, illustrated at the beginning of Section 4. Indeed, once we apply such an encoding, the ontology used in the reduction to prove Case 3 of Theorem 6.1 contains functional roles that are specialized. In fact, as shown in [4] with a more involved proof, TBox reasoning in such ontologies is EXPTIME-complete (hence as hard as TBox reasoning in much more expressive DLs [7]), and instance checking and (U)CQ query answering are PTIME-complete in data complexity.

We now analyze the cases obtained from those considered in Theorem 6.1 by allowing for conjunction of concepts in the left-hand side of inclusion assertions[14].

Theorem 6.2. *Instance checking (and hence ontology satisfiability and query answering) is* PTIME-*hard in data complexity for ontologies $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox of one of the following forms:*

1. $Cl \longrightarrow A \mid \exists P.A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A$
   Assertions in $\mathcal{T}$: $Cl \sqsubseteq Cr$.
2. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \forall P.A$
   Assertions in $\mathcal{T}$: $Cl \sqsubseteq Cr$.
3. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \exists P.A$
   Assertions in $\mathcal{T}$: $Cl \sqsubseteq Cr$, $(\mathsf{funct}\ P)$.

Proof. For Case 1, the proof is by a LOGSPACE reduction of Path System Accessibility, which is PTIME-complete [40]. An instance of *Path System Accessibility* is defined as $PS = (V, E, S, t)$, where V is a set of vertexes, $E \subseteq V \times V \times V$ is an accessibility relation (we call its elements edges), $S \subseteq V$ is a set of source vertexes, and $t \in V$ is a terminal vertex. PS consists in verifying whether t is *accessible*, where accessibility is defined inductively as follows:

- each vertex $v \in S$ is accessible;
- if vertexes v_1 and v_2 are accessible and $(v, v_1, v_2) \in E$, then v is accessible;
- nothing else is accessible.

Given PS, we define the ontology $\mathcal{O} = \langle \mathcal{T}_{psa}, \mathcal{A}_{PS} \rangle$, where the TBox $\mathcal{T}_{psa}$ is constituted by the inclusion assertions

$$\exists P_1.A \sqsubseteq A_1 \qquad \exists P_2.A \sqsubseteq A_2 \qquad A_1 \sqcap A_2 \sqsubseteq A \qquad \exists P_3.A \sqsubseteq A$$

and the ABox $\mathcal{A}_{PS}$ makes use of the vertexes in V and the edges in E as constants, as described below. Consider a vertex $v \in V$, and let $e_1, \dots, e_k$ be all edges in E that have v as their first component, taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_3(v, e_1)$, and $P_3(e_i, e_{i+1})$ for $i \in \{1, \dots, k-1\}$,
- $P_1(e_i, v_1)$ and $P_2(e_i, v_2)$, where $e_i = (v, v_1, v_2)$, for $i \in \{1, \dots, k-1\}$.

[14] Note that allowing for conjunction of concepts in the right-hand side of inclusion assertions does not have any impact on expressivity or complexity, since an assertion $B \sqsubseteq C_1 \sqcap C_2$ is equivalent to the pair of assertions $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$.

Additionally, $\mathcal{A}_{PS}$ contains one membership assertion $A(v)$ for each vertex $v \in S$. Again, $\mathcal{T}_{psa}$ does not depend on PS, and it is easy to see that $\mathcal{A}_{PS}$ can be constructed in LOGSPACE from PS. We show that t is accessible in PS if and only if $\mathcal{O} \models A(t)$.

"$\Leftarrow$" Suppose that t is not accessible in PS. We construct a model $\mathcal{I}$ of $\mathcal{O}$ such that $t^\mathcal{I} \notin A^\mathcal{I}$. Consider the interpretation $\mathcal{I}$ with $\Delta^\mathcal{I} = V \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^\mathcal{I}$, $P_2^\mathcal{I}$, and $P_3^\mathcal{I}$ consist of all pairs of nodes directly required by the ABox assertions, $A_1^\mathcal{I}$ consists of all edges (v', v_1, v_2) such that v_1 is accessible in PS, $A_2^\mathcal{I}$ consists of all edges (v', v_1, v_2) such that v_2 is accessible in PS, and $A^\mathcal{I}$ consists of all vertexes v that are accessible in PS union all edges (v', v_1, v_2) such that both v_1 and v_2 are accessible in PS. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{O}$, and since t is not accessible in PS, we have that $t \notin A^\mathcal{I}$.

"$\Rightarrow$" Suppose that t is accessible in PS. We prove by induction on the structure of the derivation of accessibility that if a vertex v is accessible, then $\mathcal{O} \models A(v)$. Base case (direct derivation): $v \in S$, hence, by definition, $\mathcal{A}$ contains the assertion $A(v)$ and $\mathcal{O} \models A(v)$. Inductive case (indirect derivation): there exists an edge $(v, v_1, v_2) \in E$ and both v_1 and v_2 are accessible. By the inductive hypothesis, we have that $\mathcal{O} \models A(v_1)$ and $\mathcal{O} \models A(v_2)$. Let $e_1, \ldots, e_h$ be the edges in E that have v as their first component, up to $e_h = (v, v_1, v_2)$ and in the same order used in the construction of the ABox. Then, by $P_1(e_h, v_1)$ in the ABox and the assertions $\exists P_1.A \sqsubseteq A_1$ we have that $\mathcal{O} \models A_1(e_h)$. Similarly, we get $\mathcal{O} \models A_2(e_h)$, and hence $\mathcal{O} \models A(e_h)$. By exploiting assertions $P_3(e_i, e_{i+i})$ in the ABox, and the TBox assertion $\exists P_3.A \sqsubseteq A$, we obtain by induction on h that $\mathcal{O} \models A(e_1)$. Finally, by $P_3(v, e_1)$, we obtain that $\mathcal{O} \models A(v)$.

For Cases 2 and 3, the proof follows from Case 1 and observations analogous to the ones for Theorem 6.1. □

We also state, without a proof the following result, which shows that qualified existential restrictions on the left-hand side of inclusion assertions together with inverse roles are sufficient to obtain PTIME-hardness.

Theorem 6.3. *Instance checking (and hence ontology satisfiability and query answering) is* PTIME-*hard in data complexity for ontologies* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *where* $\mathcal{A}$ *is an ABox, and* $\mathcal{T}$ *is a TBox of the form:*

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists P.A \mid \exists P^-A \\ Cr & \longrightarrow & A \mid \exists P \\ \multicolumn{3}{l}{\textit{Assertions in } \mathcal{T}\textit{:} \quad Cl \sqsubseteq Cr.} \end{array}$$

We now show three cases where the TBox language becomes so expressive that the data complexity of answering CQs becomes coNP-hard, i.e., as hard as for very expressive DLs [71].

Theorem 6.4. *Answering CQs is coNP-hard in data complexity for ontologies* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *where* $\mathcal{A}$ *is an ABox, and* $\mathcal{T}$ *is a TBox of one of the following forms:*

1. $$\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid A_1 \sqcup A_2 \\ R & \rightarrow & P \\ \multicolumn{3}{l}{\textit{Assertions in } \mathcal{T}\textit{:} \quad Cl \sqsubseteq Cr.} \end{array}$$

2. $Cl \;\rightarrow\; A \mid \neg A$
   $Cr \;\rightarrow\; A$
   Assertions in $\mathcal{T}$: $Cl \sqsubseteq Cr$.
3. $Cl \;\rightarrow\; A \mid \forall P.A$
   $Cr \;\rightarrow\; A$
   Assertions in $\mathcal{T}$: $Cl \sqsubseteq Cr$.

Proof. In all three cases, the proof is an adaptation of the proof of coNP-hardness of instance checking for the DL $\mathcal{ALE}$ presented in [39]. The proof is based on a reduction of $2+2$-CNF unsatisfiability, shown to be coNP-complete in [39], to CQ answering. A $2+2$-CNF formula on an alphabet $\mathcal{P} = \{\ell_1, \ldots, \ell_m\}$ is a CNF formula $F = C_1 \wedge \cdots \wedge C_n$ in which each clause $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$ has exactly four literals, two positive ones, L^i_{1+} and L^i_{2+}, and two negative ones, $\neg L^i_{1-}$ and $\neg L^i_{2-}$, where the propositional letters L^i_{1+}, L^i_{2+}, L^i_{1-}, and L^i_{2-} are elements of $\mathcal{P} \cup \{\mathit{true}, \mathit{false}\}$.

We first consider Case 1. Given a $2+2$-CNF formula F as above, we associate with it an ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$ and a boolean CQ q as follows. $\mathcal{O}_F$ has one constant ℓ for each letter $\ell \in \mathcal{P}$, one constant c_i for each clause C_i, plus two constants true and false for the corresponding propositional constants. The atomic concepts of $\mathcal{O}_F$ are O, A_t, and A_f, and the atomic roles are P_1, P_2, N_1, N_2. Then, we set

$$\begin{aligned}
\mathcal{T} &= \{\; O \;\sqsubseteq A_t \sqcup A_f \;\},\\
\mathcal{A}_F &= \{\; A_t(\mathit{true}),\ A_f(\mathit{false}),\ O(\ell_1),\ \ldots\ O(\ell_m),\\
&\qquad P_1(c_1, \ell^1_{1+}),\ P_2(c_1, \ell^1_{2+}),\ N_1(c_1, \ell^1_{1-}),\ N_2(c_1, \ell^1_{2-}),\\
&\qquad \cdots\\
&\qquad P_1(c_n, \ell^n_{1+}),\ P_2(c_n, \ell^n_{2+}),\ N_1(c_n, \ell^n_{1-}),\ N_2(c_n, \ell^n_{2-})\;\},\ \text{and}\\
q() &= P_1(c, f_1), A_f(f_1),\ P_2(c, f_2), A_f(f_2),\ N_1(c, t_1), A_t(t_1),\ N_2(c, t_2), A_t(t_2).
\end{aligned}$$

Notice that only the ABox $\mathcal{A}_F$ depends on the formula F, and that the TBox contains a single inclusion assertion involving a concept disjunction.

Intuitively, the membership to the extension of A_f or A_t corresponds to the truth values true and false respectively and checking whether $() \in \mathit{cert}(q, \mathcal{O}_F)$ (i.e., the query evaluates to true in $\mathcal{O}_F$) corresponds to checking whether in every truth assignment for the formula F there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e., a clause that is not satisfied. Note that the ABox $\mathcal{A}_F$ contains the assertions $A_t(\mathit{true})$ and $A_f(\mathit{false})$ in order to guarantee that in each model $\mathcal{I}$ of $\mathcal{O}_F$ the constants true and false are respectively in $A^{\mathcal{I}}_t$ and $A^{\mathcal{I}}_f$ (and possibly in both).

Now, it remains to prove that the formula F is unsatisfiable if and only if $() \in \mathit{cert}(q, \mathcal{O}_F)$.

"$\Rightarrow$" Suppose that F is unsatisfiable. Consider a model $\mathcal{I}$ of $\mathcal{O}_F$ (which always exists since $\mathcal{O}_F$ is always satisfiable), and let $\delta_{\mathcal{I}}$ be the truth assignment for F such that $\delta_{\mathcal{I}}(\ell) = \mathit{true}$ iff $\ell^{\mathcal{I}} \in A^{\mathcal{I}}_t$, for every letter $\ell \in \mathcal{P}$ (and corresponding constant in $\mathcal{O}_F$). Since F is unsatisfiable, there exists a clause C_i that is not satisfied by $\delta_{\mathcal{I}}$, and therefore $\delta_{\mathcal{I}}(L^i_{1+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{2+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{1-}) = \mathit{true}$ and $\delta_{\mathcal{I}}(L^i_{2-}) = \mathit{true}$. It follows that in $\mathcal{I}$ the interpretation of the constants related in $\mathcal{A}_F$ to c_i through the roles P_1 and P_2 is not in $A^{\mathcal{I}}_t$ and, since $\mathcal{I}$ satisfies $O \sqsubseteq A_t \sqcup A_f$, it is in $A^{\mathcal{I}}_f$. Similarly, the interpretation of the constants related to c_i through the roles N_1 and N_2 is in $A_t^{\mathcal{I}}$. Thus, there exists a substitution σ that assigns the variables in q to elements of $\Delta^{\mathcal{I}}$ in such a way that $\sigma(q)$ evaluates to true in $\mathcal{I}$ (notice that this holds even if the propositional constants *true* or *false* occur in F). Therefore, since this argument holds for each model $\mathcal{I}$ of $\mathcal{O}_F$, we can conclude that $() \in \mathit{cert}(q, \mathcal{O}_F)$.

"$\Leftarrow$" Suppose that F is satisfiable, and let δ be a truth assignment satisfying F. Let $\mathcal{I}_\delta$ be the interpretation for $\mathcal{O}_F$ defined as follows:

$$\begin{aligned}
O^{\mathcal{I}_\delta} &= \{\, \ell^{\mathcal{I}_\delta} \mid \ell \text{ occurs in } F \,\},\\
A_t^{\mathcal{I}_\delta} &= \{\, \ell^{\mathcal{I}_\delta} \mid \delta(\ell) = \mathit{true} \,\} \cup \{\mathit{true}\},\\
A_f^{\mathcal{I}_\delta} &= \{\, \ell^{\mathcal{I}_\delta} \mid \delta(\ell) = \mathit{false} \,\} \cup \{\mathit{false}\},\\
P^{\mathcal{I}_\delta} &= \{\, (a_1^{\mathcal{I}_\delta}, a_2^{\mathcal{I}_\delta}) \mid P(a_1, a_2) \in \mathcal{A}_F\}, \text{ for } P \in \{P_1, P_2, N_1, N_2\}.
\end{aligned}$$

It is easy to see that $\mathcal{I}_\delta$ is a model of $\mathcal{O}_F$. On the other hand, since δ satisfies F, for every clause c_i in F there exists a positive literal ℓ_+^i such that $\delta(\ell_+^i) = \mathit{true}$, or a negative literal ℓ_-^i such that $\delta(\ell_-^i) = \mathit{false}$. It follows that for every constant c_i, there exists either a role (P_1 or P_2) that relates c_i to a constant whose interpretation is in $A_t^{\mathcal{I}_\delta}$ or there exists a role (N_1 or N_2) that relates c_i to a constant whose interpretation is in $A_f^{\mathcal{I}_\delta}$. Since the query q evaluates to true in $\mathcal{I}_\delta$ only if there exists a constant c_i in $\mathcal{O}_F$ such that the interpretations of the constants related to c_i by roles P_1 and P_2 are both in $A_f^{\mathcal{I}_\delta}$ and the interpretations of the constants related to c_i by roles N_1 and N_2 are both in $A_t^{\mathcal{I}_\delta}$, it follows that the query q evaluates to *false* in $\mathcal{I}_\delta$ and therefore $() \notin \mathit{cert}(q, \mathcal{O}_F)$.

The proofs for Case 2 and Case 3 are obtained by reductions of $2+2$-CNF unsatisfiability to CQ answering analogous to the one for Case 1. More precisely, for Case 2 the ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$ has the same constants and the same atomic roles as for Case 1, and has only the atomic concepts A_t and A_f. Then, $\mathcal{T}_F = \{\neg A_t \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 1 but without the assertions involving the concept O. The query q is as for Case 1.

For Case 3, $\mathcal{O}_F$ has the same constants as for Cases 1 and 2, the same atomic roles as for Cases 1 and 2 plus an atomic role P_t, and two atomic concepts A and A_f. Then, $\mathcal{T} = \{\forall P_t.A \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 2 but without the assertion $A_t(\mathit{true})$, which is substituted by the assertion $P_t(\mathit{true}, a)$, where a is a new constant not occurring elsewhere in $\mathcal{O}_F$. The query is

$$\begin{aligned}
q() = {} & P_1(c, f_1), A_f(f_1), \quad && P_2(c, f_2), A_f(f_2),\\
& N_1(c, t_1), P_t(t_1, x_1), && N_2(c, t_2), P_t(t_2, x_2).
\end{aligned}$$

The correctness of the above reductions can be proved as done for Case 1. We finally point out that the intuition behind the above results is that in all three cases it is possible to require a reasoning by case analysis, caused by set covering assertions. Indeed, in Case 2 we have explicitly asserted $O \sqsubseteq A_t \sqcup A_f$, while in Case 1 and Case 3, A_t and A_f, and $\forall P_t.A$ and $\exists P_t$ cover the entire domain, respectively. $\square$

The results proved in Theorems 6.1, 6.2, 6.3, and 6.4 are summarized in Table 1. Notice that, while the NLOGSPACE-hardness and PTIME-hardness results in the table hold already for instance checking (i.e., answering atomic queries), the coNP-hardness results

Table 1. Data Complexity of query answering for various extensions of *DL-Lite*$_{\mathcal{A},id}$

Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity of query answering	Proved in
DL-Lite$_{\mathcal{A},id}$		$\surd$	$\surd^*$	in AC^0	Theorems 5.17, 5.25
$A \mid \exists P.A$	A	–	–	NLOGSPACE-hard	Theorem 6.1, Case 1
A	$A \mid \forall P.A$	–	–	NLOGSPACE-hard	Theorem 6.1, Case 2
A	$A \mid \exists P.A$	$\surd$	–	NLOGSPACE-hard	Theorem 6.1, Case 3
$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTIME-hard	Theorem 6.2, Case 1
$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTIME-hard	Theorem 6.2, Case 2
$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTIME-hard	Theorem 6.2, Case 3
$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTIME-hard	Theorem 6.3
A	$A \mid A_1 \sqcup A_2$	–	–	coNP-hard	Theorem 6.4, Case 1
$A \mid \neg A$	A	–	–	coNP-hard	Theorem 6.4, Case 2
$A \mid \forall P.A$	A	–	–	coNP-hard	Theorem 6.4, Case 3

Legenda: A (possibly with subscript) = atomic concept, P = atomic role,
Cl/*Cr* = left/right-hand side of inclusion assertions, $\mathcal{F}$ = functionality assertions allowed,
$\mathcal{R}$ = role/relationship inclusions allowed, where * denotes restricted interaction between functionality and role inclusion, according to Definition 2.1.
The NLOGSPACE and PTIME hardness results hold already for instance checking.

proved in Theorem 6.4 hold for CQ answering, but do *not* hold for instance checking. Indeed, as shown in [4], instance checking (and hence ontology satisfiability) stays in AC^0 in data complexity for *DL-Lite*$_{\mathcal{A}}$ extended with arbitrary boolean combinations (i.e., negation, and disjunction) of concepts, both in the left-hand side and in the right-hand side of inclusion assertions. [4] shows also that *DL-Lite*$_{\mathcal{A}}$ can be extended with *number restrictions* (cf. also Section 2.2) and with the additional role constraints present in OWL 2 QL that are not already expressible in *DL-Lite*$_{\mathcal{A}}$, such as reflexivity, irreflexivity, and asymmetry, without losing FOL-rewritability of satisfiability and UCQ query answering.

Finally, the following result from [25] motivates the locality restriction in identification assertions, i.e., that at least one of the paths in an identification assertion must have length 1. Indeed, if such a restriction is removed, we lose again FOL-rewritability of reasoning.

Theorem 6.5. *Ontology satisfiability (and hence instance checking and query answering) in DL-Lite*$_{\mathcal{A}}$ *extended with single-path IdCs that are non-local is* NLOGSPACE-*hard in data complexity.*

Proof. The proof is based again on a reduction of reachability in directed graphs (see proof of Theorem 6.1) to ontology satisfiability. Let $G = \langle V, E\rangle$ be a directed graph, where V is a set of vertexes and E a set of directed edges, and let s and t be two vertexes of G. We consider the graph represented through functional relations F (to connect a vertex to the first element of the chain of its children), N (to connect an element of the chain to the next), and S (to connect the elements forming the chain to the actual

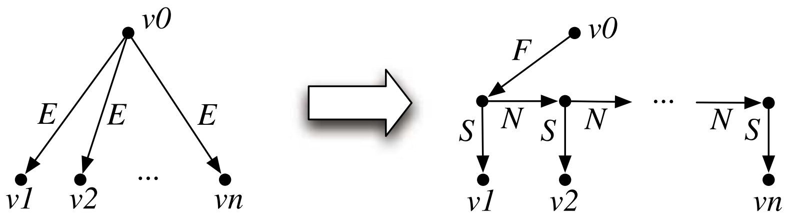


Fig. 17. Representation of a graph through the functional relations F, N, S

child vertexes of the graph), and denote with V^+ the set of vertexes augmented by the vertexes used in such a representation (cf. Figure 17).

From G and the two vertexes s and t, we define the ontology $\mathcal{O}_{idcs} = \langle \mathcal{T}_{idcs}, \mathcal{A}_G \rangle$ as follows:

– The alphabet of $\mathcal{T}_{idcs}$ consists of an atomic concept A, that intuitively denotes the vertexes of two copies of G, of an atomic concept A_t, and of atomic roles P_F, P_N, P_S, and P_0. Then

$$\mathcal{T}_{idcs} \;=\; \{A \sqsubseteq \exists P_0,\ (\mathsf{id}\ A_t\ P_0)\} \cup \{(\mathsf{id}\ \exists P_0^-\ P_0^- \circ P_{\mathcal{R}}^- \circ P_0) \mid \mathcal{R} \in \{F, N, S\}\}.$$

Notice that $\mathcal{T}_{idcs}$ does not depend on G.

– The ABox $\mathcal{A}_G$ is defined from the graph G and the two vertexes s and t as follows:

$$\begin{aligned}\mathcal{A}_G \;=\; &\{P_{\mathcal{R}}(a_1, a_2),\ P_{\mathcal{R}}(a_1', a_2') \mid (a_1, a_2) \in \mathcal{R},\ \text{for } \mathcal{R} \in \{F, N, S\}\} \cup \\ &\{A(a),\ A(a') \mid a \in V^+\} \cup \{P_0(s, a_{init}),\ P_0(s', a_{init}),\ A_t(t),\ A_t(t')\}.\end{aligned}$$

In other words, we introduce for each node a of the graph G two constants a and a' in $\mathcal{O}$, and we encode in $\mathcal{A}_G$ two copies of (the representation of) G. In addition, we include in $\mathcal{A}_G$ the assertions $P_0(s, a_{init})$ and $P_0(s', a_{init})$ connecting the two copies of the start vertex s to an additional constant a_{init} that does not correspond to any vertex of (the representation of) G. We also include the assertions $A_t(t)$ and $A_t(t')$, which are exploited to encode the reachability test (cf. Figure 18).

It can be shown by induction on the length of paths from s, that t is reachable from s in G iff $\mathcal{O}_{idcs}$ is unsatisfiable. Intuitively, the TBox enforces that each individual contributing to the encoding of the two copies of G has an outgoing P_0 edge. Moreover, the path-identification assertions enforce that each object that is in the range of such a P_0 edge is identified by a suitable path. Hence, starting from a_{init}, corresponding pairs of objects (in the range of P_0) in the two copies of the graph that are reachable from s and s', respectively, will be unified with each other. If t is reachable from s, also the two objects connected to t and t' via P_0 will be unified. Hence by the identification assertion $(\mathsf{id}\ A_t\ P_0)$, we have that t and t' are forced to be equal, which makes the ontology unsatisfiable (due to the unique name assumption). Notice that, for the reduction to work, we needed to make sure that each vertex has at most one outgoing edge, hence we have preliminarily encoded the edge relation E using the functional relations F, N, and S. $\square$

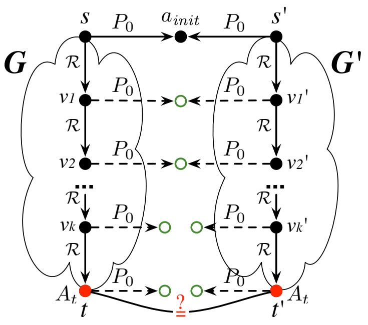


Fig. 18. Structure of a potential model of the ontology $\mathcal{O}_{idcs}$ used in the proof of Theorem 6.5

6.2 Changing the *DL-Lite* Semantics

Concerning the possibility of strengthening the semantics, we analyze the consequences of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of an ontology, two distinct constants denote two different domain elements. Unfortunately, this leads instance checking (and satisfiability) out of AC^0, and therefore instance checking and query answering are not FOL-rewritable anymore.

Theorem 6.6. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology interpreted without the unique name assumption. Then instance checking with respect to $\mathcal{O}$ is* NLOGSPACE-*hard in the size of $\mathcal{A}$.*

Proof. The proof is based again on a LOGSPACE reduction of reachability in directed graphs to instance checking. Let $G = \langle V, E \rangle$ be a directed graph and s and t two vertexes of G. As in the proof of Theorem 6.5, we consider G represented through first-child and next-sibling functional relations F, N, S (cf. Figure 17).

From G and the two vertexes s and t, we define an ontology $\mathcal{O}_{una} = \langle \mathcal{T}_{una}, \mathcal{A}_G \rangle$ as follows:

– The alphabet of $\mathcal{T}_{una}$ consists of an atomic concept A and of atomic roles P_F, P_N, P_S, and P_0. The TBox itself imposes only that all roles are functional, i.e.,

$$\mathcal{T}_{una} = \{(\mathsf{funct}\ P_0)\} \cup \{(\mathsf{funct}\ P_\mathcal{R}) \mid \mathcal{R} \in \{F, N, S\}\}.$$

Notice that $\mathcal{T}_{una}$ does not depend on G.

– The ABox $\mathcal{A}_G$ is defined from the graph G and the two vertexes s and t as follows:

$$\begin{aligned}\mathcal{A}_G = \ & \{P_\mathcal{R}(a_1, a_2), P_\mathcal{R}(a'_1, a'_2) \mid (a_1, a_2) \in \mathcal{R},\ \text{for } \mathcal{R} \in \{F, N, S\}\} \cup \\ & \{A(t),\ P_0(a_{init}, s),\ P_0(a_{init}, s')\}\end{aligned}$$

In other words, we introduce for each node a of the graph G two constants a and a' in $\mathcal{O}$, and we encode in $\mathcal{A}_G$ two copies of (the representation of) G. In addition, we include in $\mathcal{A}_G$ the facts $P_0(a_{init}, s)$, $P_0(a_{init}, s')$, and $A(t)$, where a_{init} is an additional constant that does not correspond to any vertex of (the representation of) G (cf. Figure 19).

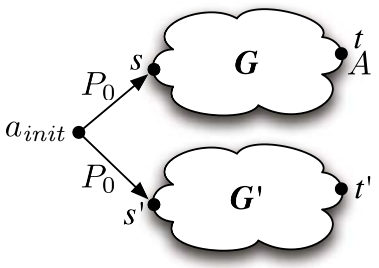


Fig. 19. Structure of the ABox $\mathcal{A}_G$ used in the proof of Theorem 6.6

It is now possible to prove that t is reachable from s in G if and only if $\mathcal{O}_{una} \models A(t')$. Indeed, it is easy to verify that the latter holds if and only if for every model $\mathcal{I}$ of $\mathcal{O}_{una}$, the constants t and t' are interpreted as the same object, i.e., $t^{\mathcal{I}} = t'^{\mathcal{I}}$. This is the case if and only if $t^{\mathcal{I}}$ and $t'^{\mathcal{I}}$ are forced to be equal by the functionality of the roles P_0, P_F, P_N, and P_S. By exploiting the structure of the ABox $\mathcal{A}_G$, one can prove by induction on the length of paths from s, that such an equality is enforced if and only if t is reachable from s in G. □

7 Accessing Data through *DL-Lite*$_{\mathcal{A},id}$ Ontologies

The discussion presented in the previous sections on *DL-Lite*$_{\mathcal{A},id}$ ontologies assumed a relational representation for the ABox assertions. This is a reasonable assumption only in those cases where the ontology is managed by an ad hoc system, and is built from scratch for the specific application.

We argue that this is not a typical scenario in current applications (e.g., in Enterprise Application Integration). Indeed, we believe that one of the most interesting real-world usages of ontologies is what we have called *ontology-based data access* (OBDA). OBDA is the problem of accessing a set of existing data sources by means of a conceptual representation expressed in terms of an ontology. In such a scenario, the TBox of the ontology provides a shared, uniform, abstract view of the intensional level of the application domain, whereas the information about the extensional level (the instances of the ontology) resides in the data sources, which are developed independently of the conceptual layer, and are managed by traditional technologies (such as relational database technology). In other words, the ABox of the ontology does not exist as an independent syntactic object. Rather, the instances of concepts and roles in the ontology are simply an abstract and virtual representation of some real data stored in existing data sources. Therefore, the problem arises of establishing sound mechanisms for linking existing data to the instances of the concepts and the roles in the ontology.

In this section, we present a solution that has been proposed recently for this problem [75], based on a mapping mechanism that enables a designer to link existing data sources to an ontology expressed in *DL-Lite*$_{\mathcal{A},id}$, and by illustrating a formal framework capturing the notion of *DL-Lite*$_{\mathcal{A},id}$ *ontology with mappings*. In the following, we assume that the data sources are expressed in terms of the relational data model. In other words, all the technical development presented in the rest of this section assumes that the set of sources to be linked to the ontology constitutes a single relational database. Note that this is a realistic assumption, since many data federation tools are now available that are able to wrap a set of heterogeneous sources and present them as a single relational database.

Before delving into the details of the method, a preliminary discussion on the notorious *impedance mismatch problem* between values (data) and objects is in order [66]. When mapping relational data sources to ontologies, one should take into account that sources store values, whereas instances of concepts are objects, where each object should be denoted by an ad hoc identifier (e.g., a constant in logic), not to be confused with any data item. For example, if a data source stores data about persons, it is likely that values for social security numbers, names, etc. will appear in the sources. However, at the conceptual level, the ontology will represent persons in terms of a concept, and instances of such concepts will be denoted by object constants.

One could argue that data sources might, in some cases, store directly object identifiers. However, in order to use such object identifiers at the conceptual level, one should make sure that such identifiers have been chosen on the basis of an "agreement" among the sources on the form used to represent objects. This is something occurring very rarely in practice. For all the above reasons, in *DL-Lite*$_{\mathcal{A},id}$, we take a radical approach. To face the impedance mismatch problem, and to tackle the possible lack of an a-priori agreement on identification mechanisms at the sources, we keep data values appearing in the sources separate from object identifiers at the conceptual level. In particular, we consider object identifiers formed by (logic) terms built out of data values stored at the sources. The way by which these terms will be defined starting from the data at the sources will be specified through suitable mapping assertions, to be described below. Note that this idea traces back to the work done in deductive object-oriented databases [54].

7.1 Linking Relational Data to Ontologies

To realize the above described idea from a technical point of view, we specialize the alphabets of object constants in a particular way, which we now describe in detail.

We remind the reader that Γ_V is the alphabet of value constants in *DL-Lite*$_{\mathcal{A},id}$. We assume that data appearing at the sources are denoted by constants in Γ_V[15], and we introduce a new alphabet Λ of *function symbols*, where each function symbol has an associated *arity*, specifying the number of arguments it accepts. On the basis of Γ_V and Λ, we inductively define the set $\tau(\Lambda, \Gamma_V)$ of all *object terms* (or simply, *terms*) of the form $\mathbf{f}(d_1, \ldots, d_n)$ such that

- $\mathbf{f} \in \Lambda$,
- the arity of $\mathbf{f}$ is $n > 0$, and
- $d_1, \ldots, d_n \in \Gamma_V$.

[15] We could also introduce suitable conversion functions in order to translate values stored at the sources into value constants in Γ_V, but, for the sake of simplicity, we do not deal with this aspect here.

We finally sanction that the set Γ_O of symbols used in *DL-Lite*$_{\mathcal{A},id}$ for denoting objects actually coincides with $\tau(\Lambda, \Gamma_V)$. In other words, we use the terms built from Γ_V using the function symbols in Λ for denoting the instances of concepts in ontologies.

All the notions defined for our logics remain unchanged. In particular, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ still assigns a different element of $\Delta^{\mathcal{I}}$ to every element of Γ, and, given that Γ_O coincides with $\tau(\Lambda, \Gamma_V)$, this implies that different terms in $\tau(\Lambda, \Gamma_V)$ are interpreted as different objects in $\Delta^{\mathcal{I}}_O$, i.e., we enforce the unique name assumption on terms. Formally, this means that $\mathcal{I}$ is such that

- for each $a \in \Gamma_V$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_V$,
- for each $a \in \Gamma_O$, i.e., for each $a \in \tau(\Lambda, \Gamma_V)$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_O$,
- for each $a_1, a_2 \in \Gamma$, $a_1 \neq a_2$ implies $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$.

The syntax and the semantics of a *DL-Lite*$_{\mathcal{A}}$ TBox, ABox, and UCQ, introduced in Section 2, do not need to be modified. In particular, from the point of view of the semantics of queries, the notion of certain answers is exactly the same as the one presented in Section 2.4.

We can now turn our attention to the problem of specifying mapping assertions linking the data at the sources to the objects in the ontology. As mentioned, we assume that the data sources are wrapped into a relational database $\mathcal{D}$ (constituted by the relational schema, and the extensions of the relations), so that we can query such data by using SQL, and that all value constants stored in $\mathcal{D}$ belong to Γ_V Also, the database $\mathcal{D}$ is independent from the ontology; in other words, our aim is to link to the ontology a collection of data that exist autonomously, and have not been necessarily structured with the purpose of storing the ontology instances.

In the following, we denote with $ans(\varphi, \mathcal{D})$ the set of tuples (of the arity of φ) of value constants returned as the result of the evaluation of the SQL query φ over the database $\mathcal{D}$.

With these assumptions in place, to actually realize the link between the data and the ontology, we adapt principles and techniques from the literature on data integration [63]. In particular, we resort to *mappings* as described in the following definition. We make use of the notion of *variable term*, which is a term of the same form as the object terms introduced above, with the difference that variables may appear as arguments of the function. In other words, a variable term has the form $\mathbf{f}(\boldsymbol{z})$, where $\mathbf{f}$ is a function symbol in Λ of arity m, and $\boldsymbol{z}$ denotes an m-tuple of variables or value constants.

Definition 7.1. *A DL-Lite*$_{\mathcal{A},id}$ ontology with mappings *is a triple* $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, *where:*

- $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{A},id}$ *TBox;*
- $\mathcal{D}$ *is a relational database;*
- $\mathcal{M}$ *is a set of* mapping assertions, *partitioned into two sets,* $\mathcal{M}^t$ *and* $\mathcal{M}^a$, *where:*
  - $\mathcal{M}^t$ *is a set of so-called* typing mapping assertions, *each one of the form*

$$\Phi \rightsquigarrow T_i,$$

where Φ *is a query of arity 1 over* $\mathcal{D}$, *denoting the projection of one relation over one of its columns, and* T_i *is one of the DL-Lite*$_{\mathcal{A},id}$ *data types;*

- $\mathcal{M}^a$ *is a set of* data-to-object mapping assertions *(or simply mapping assertions), each one of the form*

$$\Phi(\boldsymbol{x}) \rightsquigarrow \Psi(\boldsymbol{y}, \boldsymbol{t}),$$

where

 * $\boldsymbol{x}$ *is a non-empty set of variables,*
 * $\boldsymbol{y} \subseteq \boldsymbol{x}$*,*
 * $\boldsymbol{t}$ *is a set of variable terms of the form* $\boldsymbol{f}(\boldsymbol{z})$*, with* $\boldsymbol{f} \in \Lambda$ *and* $\boldsymbol{z} \subseteq \boldsymbol{x}$*,*
 * $\Phi(\boldsymbol{x})$ *is an arbitrary SQL query over* $\mathcal{D}$*, with* $\boldsymbol{x}$ *as output variables, and*
 * $\Psi(\boldsymbol{y}, \boldsymbol{t})$ *is a CQ over* $\mathcal{T}$ *of arity* $n > 0$ *without non-distinguished variables, whose atoms are over the variables* $\boldsymbol{y}$ *and the variable terms* $\boldsymbol{t}$*.*

We briefly comment on the assertions in $\mathcal{M}$ as defined above. Typing mapping assertions are used to assign appropriate types to constants in the relations of $\mathcal{D}$. Basically, these assertions are used for interpreting the values stored in the database in terms of the types used in the ontology, and their usefulness is evident in all cases where the types in the data sources do not directly correspond to the types used in the ontology. Data-to-object mapping assertions, on the other hand, are used to map data in the database to instances of concepts, roles, and attributes in the ontology.

We next give an example of *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings.

Example 7.2. Let $\mathcal{D}_{pr}$ be the database constituted by a set of relations with the following signature:

$$\begin{array}{l} D_1[\texttt{SSN}:\textbf{STRING},\ \texttt{PROJ}:\textbf{STRING},\ \texttt{D}:\textbf{DATE}],\\ D_2[\texttt{SSN}:\textbf{STRING},\ \texttt{NAME}:\textbf{STRING}],\\ D_3[\texttt{CODE}:\textbf{STRING},\ \texttt{NAME}:\textbf{STRING}],\\ D_4[\texttt{CODE}:\textbf{STRING},\ \texttt{SSN}:\textbf{STRING}] \end{array}$$

We assume that, from the analysis of the above data sources, the following meaning of the above relations has been derived.

- Relation D_1 stores tuples (s, p, d), where s and p are strings and d is a date, such that s is the social security number of a temporary employee, p is the name of the project she works for (different projects have different names), and d is the ending date of the employment.
- Relation D_2 stores tuples (s, n) of strings consisting of the social security number s of an employee and her name n.
- Relation D_3 stores tuples (c, n) of strings consisting of the code c of a manager and her name n.
- Finally, relation D_4 relates managers' code with their social security number.

A possible extension for the above relations is given by the following sets of tuples:

$$\begin{array}{l} D_1 = \{(\texttt{20903},\texttt{"Tones"},\texttt{25/03/09})\}\\ D_2 = \{(\texttt{20903},\texttt{"Rossi"}),(\texttt{55577},\texttt{"White"})\}\\ D_3 = \{(\texttt{"X11"},\texttt{"White"}),(\texttt{"X12"},\texttt{"Black"})\}\\ D_4 = \{(\texttt{"X11"},\texttt{29767})\} \end{array}$$

$$\begin{array}{rclrcl}
\mathit{Manager} & \sqsubseteq & \mathit{Employee} & \mathit{Project} & \sqsubseteq & \delta(\mathbf{projName}) \\
\mathit{TempEmp} & \sqsubseteq & \mathit{Employee} & \rho(\mathbf{projName}) & \sqsubseteq & \mathtt{xsd{:}string} \\
\mathit{Employee} & \sqsubseteq & \mathit{Person} & & (\mathsf{funct}\ \mathbf{projName}) & \\
\mathit{Employee} & \sqsubseteq & \exists \mathit{WORKS\text{-}FOR} & \mathit{TempEmp} & \sqsubseteq & \delta(\mathbf{until}) \\
\exists \mathit{WORKS\text{-}FOR}^- & \sqsubseteq & \mathit{Project} & \delta(\mathbf{until}) & \sqsubseteq & \exists \mathit{WORKS\text{-}FOR} \\
\mathit{Person} & \sqsubseteq & \delta(\mathbf{persName}) & \rho(\mathbf{until}) & \sqsubseteq & \mathtt{xsd{:}date} \\
\rho(\mathbf{persName}) & \sqsubseteq & \mathtt{xsd{:}string} & & (\mathsf{funct}\ \mathbf{until}) & \\
& (\mathsf{funct}\ \mathbf{persName}) & & \mathit{Manager} & \sqsubseteq & \neg\delta(\mathbf{until})
\end{array}$$

Fig. 20. The *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{pr}$ for the projects example

m_1^t :	SELECT SSN FROM D_1	$\rightsquigarrow$	xsd:string
m_2^t :	SELECT SSN FROM D_2	$\rightsquigarrow$	xsd:string
m_3^t :	SELECT CODE FROM D_3	$\rightsquigarrow$	xsd:string
m_4^t :	SELECT CODE FROM D_4	$\rightsquigarrow$	xsd:string
m_5^t :	SELECT PROJ FROM D_1	$\rightsquigarrow$	xsd:string
m_6^t :	SELECT NAME FROM D_2	$\rightsquigarrow$	xsd:string
m_7^t :	SELECT NAME FROM D_3	$\rightsquigarrow$	xsd:string
m_8^t :	SELECT SSN FROM D_4	$\rightsquigarrow$	xsd:string
m_9^t :	SELECT D FROM D_1	$\rightsquigarrow$	xsd:date

Fig. 21. The typing mapping assertions $\mathcal{M}_{pr}^t$ for the projects example

Consider now the TBox $\mathcal{T}_{pr}$ shown in Figure 20, which models information about employees and projects they work for. Specifically, the assertions in $\mathcal{T}_{pr}$ state the following. Managers and temporary employees are two kinds of employees, and employees are persons. Each employee works for at least one project, whereas each person and each project has a unique name. Both person names and project names are strings, whereas the attribute **until** associates objects with a unique date. In particular, any temporary employee has an associated date (which indicates the expiration date of her contract), and everyone having a value for the attribute **until** participates in the role *WORKS-FOR*. Finally, $\mathcal{T}_{pr}$ specifies that a manager does not have any value for the attribute **until**, meaning that a manager has a permanent position. Note that this implies that no employee is simultaneously a temporary employee and a manager.

Now, let $\Lambda = \{\mathbf{pers}, \mathbf{proj}, \mathbf{mgr}\}$ be a set of function symbols, all of arity 1. Consider the *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM}_{pr} = \langle \mathcal{T}_{pr}, \mathcal{M}_{pr}, \mathcal{D}_{pr} \rangle$, where $\mathcal{M}_{pr} = \mathcal{M}_{pr}^t \cup \mathcal{M}_{pr}^a$, with $\mathcal{M}_{pr}^t$ shown in Figure 21, and $\mathcal{M}_{pr}^a$ shown in Figure 22. We briefly comment on the data-to-ontology mapping assertions in $\mathcal{M}_{pr}^a$:

- m_1^a maps every tuple (s, p, d) in D_1 to a temporary employee $\mathbf{pers}(s)$, working until d for project $\mathbf{proj}(p)$ with name p.
- m_2^a maps every tuple (s, n) in D_2 to an employee $\mathbf{pers}(s)$ with name n.
- m_3^a and m_4^a tell us how to map data in D_3 and D_4 to managers and their name in the ontology. Note that, if D_4 provides the social security number s of a manager whose

code is in D_3, then we use the social security number to form the corresponding object term, i.e., the object term has the form $\mathbf{pers}(s)$. Instead, if D_4 does not provide this information, then we use an object term of the form $\mathbf{mgr}(c)$, where c is a code, to denote the corresponding instance of the concept *Manager*. ■

7.2 Semantics of Ontologies with Mappings

In order to define the semantics of a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings, we need to define when an interpretation *satisfies an assertion in $\mathcal{M}$ w.r.t. a database $\mathcal{D}$*. To this end, we make use of the notion of ground instance of a formula. Let $\Psi(\boldsymbol{x})$ be a formula over a *DL-Lite*$_{\mathcal{A},id}$ TBox with n distinguished variables $\boldsymbol{x}$, and let $\boldsymbol{v}$ be a tuple of value constants of arity n. Then the ground instance $\Psi[\boldsymbol{x}/\boldsymbol{v}]$ of $\Psi(\boldsymbol{x})$ is the formula obtained from $\Psi(\boldsymbol{x})$ by substituting every occurrence of x_i with v_i, for $i \in \{1,\ldots,n\}$. We are now ready to define when an interpretation satisfies a mapping assertion.

In the following, we denote with $ans(\varphi, \mathcal{D})$ the set of tuples (of the arity of φ) of value constants returned as the result of the evaluation of the SQL query φ over the database $\mathcal{D}$.

Definition 7.3. *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$, with $\mathcal{M} = \mathcal{M}^t \cup \mathcal{M}^a$, be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings and $\mathcal{I}$ an interpretation of $\mathcal{OM}$.*

- *Let m^t be an assertion in $\mathcal{M}^t$ of the form $\Phi \rightsquigarrow T_i$. We say that $\mathcal{I}$* satisfies *m^t* w.r.t. *$\mathcal{D}$, if for every $v \in ans(\Phi, \mathcal{D})$, we have that $v \in val(T_i)$.*
- *Let m^a be an assertion in $\mathcal{M}^a$ of the form $\Phi(\boldsymbol{x}) \rightsquigarrow \Psi(\boldsymbol{y}, \boldsymbol{t})$, where $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{t}$ are as in Definition 7.1. We say that $\mathcal{I}$* satisfies *m^a* w.r.t. *$\mathcal{D}$, if for every tuple of values $\boldsymbol{v}$ such that $\boldsymbol{v} \in ans(\Phi, \mathcal{D})$, and for every ground atom X in $\Psi[\boldsymbol{x}/\boldsymbol{v}]$, we have that:*
  - *if X has the form $A(s)$, then $s^{\mathcal{I}} \in A^{\mathcal{I}}$;*
  - *if X has the form $F(s)$, then $s^{\mathcal{I}} \in F^{\mathcal{I}}$;*
  - *if X has the form $P(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$;*
  - *if X has the form $U(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in U^{\mathcal{I}}$.*

m_1^a :	`SELECT SSN, PROJ, D` `FROM` D_1	$\rightsquigarrow$	*TempEmp*($\mathbf{pers}$(SSN)), *WORKS-FOR*($\mathbf{pers}$(SSN), $\mathbf{proj}$(PROJ)), $\mathbf{projName}$($\mathbf{proj}$(PROJ), PROJ), $\mathbf{until}$($\mathbf{pers}$(SSN), D)
m_2^a :	`SELECT SSN, NAME` `FROM` D_2	$\rightsquigarrow$	*Employee*($\mathbf{pers}$(SSN)), $\mathbf{persName}$($\mathbf{pers}$(SSN), NAME)

m_3^a :	`SELECT SSN, NAME`
	 `FROM` D_3, D_4
	 `WHERE` D_3`.CODE =` D_4`.CODE`
		$\rightsquigarrow$	*Manager*($\mathbf{pers}$(SSN)),
			 $\mathbf{persName}$($\mathbf{pers}$(SSN), NAME)

m_4^a :	`SELECT CODE, NAME`
	 `FROM` D_3
	 `WHERE CODE NOT IN`
	 `(SELECT CODE FROM` D_4`)`
		$\rightsquigarrow$	*Manager*($\mathbf{mgr}$(CODE)),
			 $\mathbf{persName}$($\mathbf{mgr}$(CODE), NAME)

Fig. 22. The object-to-data mapping assertions $\mathcal{M}^a_{pr}$ for the projects example

We say that $\mathcal{I}$ satisfies *$\mathcal{M}$ w.r.t. $\mathcal{D}$, if it satisfies every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$. We say that $\mathcal{I}$ is a* model *of $\mathcal{OM}$ if $\mathcal{I}$ is a model of $\mathcal{T}$ and satisfies $\mathcal{M}$ w.r.t. $\mathcal{D}$. Finally, we denote with $Mod(\mathcal{OM})$ the set of models of $\mathcal{OM}$, and we say that $\mathcal{OM}$ is* satisfiable *if $Mod(\mathcal{OM}) \neq \emptyset$.*

Example 7.4. One can easily verify that the ontology with mappings $\mathcal{OM}_{pr}$ of Example 7.2 is satisfiable. ■

Note that the mapping mechanism described above nicely deals with the fact that the database $\mathcal{D}$ and the ontology $\mathcal{OM}$ are based on different semantic assumptions. Indeed, the semantics of $\mathcal{D}$ follows the so-called "closed world assumption" [79], which intuitively sanctions that every fact that is not explicitly stored in the database is false. On the contrary, the semantics of $\mathcal{OM}$ is open, in the sense that nothing is assumed about the facts that do not appear explicitly in the ABox. In a mapping assertion of the form $\Phi \rightsquigarrow \Psi$, the closed semantics of $\mathcal{D}$ is taken into account by the fact that Φ is evaluated as a standard relational query over the database $\mathcal{D}$, while the open semantics of $\mathcal{OM}$ is reflected by the fact that mappings assertions are interpreted as "material implication" in logic. It is well known that a material implication of the form $\Phi \rightsquigarrow \Psi$ imposes that every tuple of Φ contributes to the answers to Ψ, leaving open the possibility of additional tuples satisfying Ψ.

Let q denote a UCQ expressed over the TBox $\mathcal{T}$ of $\mathcal{OM}$. We call *certain answers to q over $\mathcal{OM}$*, denoted $cert(q, \mathcal{OM})$, the set of n-tuples of terms in Γ, defined as

$$cert(q, \mathcal{OM}) = \{\boldsymbol{t} \mid \boldsymbol{t}^{\mathcal{I}} \in q^{\mathcal{I}}, \text{ for all } \mathcal{I} \in Mod(\mathcal{OM})\}.$$

Given an ontology with mappings and a query q over its TBox, *query answering* is the problem of computing the certain answers to q.

7.3 Satisfiability and Query Answering for Ontologies with Mappings

Our goal is to illustrate a method for checking satisfiability and for query answering for *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings. We will give here just an overview of the method, concentrating on query answering, and refer to [75] for more details.

The simplest way to tackle reasoning over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings is to use the mappings to produce an actual ABox, and then reason on the ontology constituted by the ABox and the original TBox by applying the techniques described in Sections 4 and 5. We call such an approach "bottom-up". However, the bottom-up approach requires to actually build the ABox starting from the data at the sources, thus somehow duplicating the information already present in the data sources. To avoid this redundancy, we propose an alternative approach, called "top-down", which essentially keeps the ABox virtual.

We sketch the main ideas of both approaches below. As said, we refer in particular to query answering, but similar considerations hold for satisfiability checking too. Before delving into the discussion, we define the notions of split version of an ontology and of virtual ABox, which will be useful in the sequel.

We first show how to compute the *split version* of an ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, which has a particularly "friendly form". Specifically, we denote with $split(\mathcal{M})$ a new set of mapping assertions, obtained from $\mathcal{M}$ as follows:

m^a_{11} : SELECT SSN, PROJ, D FROM D_1 $\rightsquigarrow$ *TempEmp*(**pers**(SSN))

m^a_{12} : SELECT SSN, PROJ, D FROM D_1 $\rightsquigarrow$ *WORKS-FOR*(**pers**(SSN), **proj**(PROJ))

m^a_{13} : SELECT SSN, PROJ, D FROM D_1 $\rightsquigarrow$ **projName**(**proj**(PROJ), PROJ)

m^a_{14} : SELECT SSN, PROJ, D FROM D_1 $\rightsquigarrow$ **until**(**pers**(SSN), D)

m^a_{21} : SELECT SSN, NAME FROM D_2 $\rightsquigarrow$ *Employee*(**pers**(SSN))

m^a_{22} : SELECT SSN, NAME FROM D_2 $\rightsquigarrow$ **persName**(**pers**(SSN), NAME)

m^a_{31} : SELECT SSN, NAME FROM D_3, D_4 WHERE D_3.CODE = D_4.CODE $\rightsquigarrow$ *Manager*(**pers**(SSN))

m^a_{32} : SELECT SSN, NAME FROM D_3, D_4 WHERE D_3.CODE = D_4.CODE $\rightsquigarrow$ **persName**(**pers**(SSN), NAME)

m^a_{41} : SELECT CODE, NAME FROM D_3 WHERE CODE NOT IN (SELECT CODE FROM D_4) $\rightsquigarrow$ *Manager*(**mgr**(CODE))

m^a_{42} : SELECT CODE, NAME FROM D_3 WHERE CODE NOT IN (SELECT CODE FROM D_4) $\rightsquigarrow$ **persName**(**mgr**(CODE), NAME)

Fig. 23. The split version of the object-to-data mapping assertions $\mathcal{M}^a_{pr}$ for the projects example

(1) *split*$(\mathcal{M})$ contains all typing assertions in $\mathcal{M}$.
(2) *split*$(\mathcal{M})$ contains one mapping assertion $\Phi' \rightsquigarrow X$, for each mapping assertion $\Phi \rightsquigarrow \Psi \in \mathcal{M}$ and for each atom $X \in \Psi$, where Φ' is the projection of Φ over the variables occurring in X.

We denote with *split*$(\mathcal{OM})$ the ontology $\langle \mathcal{T}, \mathit{split}(\mathcal{M}), \mathcal{D}\rangle$.

Example 7.5. Consider the ontology with mappings $\mathcal{OM}_{pr} = \langle \mathcal{T}_{pr}, \mathcal{M}_{pr}, \mathcal{D}_{pr}\rangle$ of Example 7.2. By splitting the mappings as described above, we obtain the ontology $\mathit{split}(\mathcal{OM}_{pr}) = \langle \mathcal{T}_{pr}, \mathit{split}(\mathcal{M}_{pr}), \mathcal{D}_{pr}\rangle$, where $\mathit{split}(\mathcal{M}_{pr})$ contains all typing assertions in $\mathcal{M}_{pr}$ and the split mapping assertions shown in Figure 23. ■

The relationship between an ontology with mappings and its split version is characterized by the following theorem.

Proposition 7.6. *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings. Then, we have that*

$$\mathit{Mod}(\mathit{split}(\mathcal{OM})) = \mathit{Mod}(\mathcal{OM}).$$

Proof. The result follows straightforwardly from the syntax and the semantics of the mappings. □

This result essentially tells us that every ontology with mappings is logically equivalent to the corresponding split version. Therefore, given an arbitrary *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings, we can always reduce it to its split version. Moreover, such a reduction can be computed in LOGSPACE in the size of the mappings and does not depend on the size of the data. Therefore, in the following, we will to deal only with split versions of *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings.

In order to express the semantics of ontologies with mappings in terms of the semantics of conventional ontologies, we introduce now the notion of virtual ABox. Intuitively, given a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$, the virtual ABox corresponding to $\mathcal{OM}$ is the ABox whose assertions are computed by "applying" the mapping assertions in $\mathcal{M}$ starting from the data in $\mathcal{D}$. Note that in our method this ABox is "virtual", in the sense that it is not explicitly built.

Definition 7.7. *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings, and let $m = \Phi(\boldsymbol{x}) \rightsquigarrow X(\boldsymbol{y}, \boldsymbol{t})$ be a (split) mapping assertion in $\mathcal{M}$. The* virtual ABox generated by m from $\mathcal{D}$ *is the set of membership assertions*

$$\mathcal{A}(m, \mathcal{D}) = \{X[\boldsymbol{x}/\boldsymbol{v}] \mid \boldsymbol{v} \in ans(\Phi, \mathcal{D})\}.$$

Moreover, the virtual ABox for $\mathcal{OM}$, *denoted $\mathcal{A}(\mathcal{M}, \mathcal{D})$, is the set of membership assertions*

$$\mathcal{A}(\mathcal{M}, \mathcal{D}) = \bigcup_{m \in \mathcal{M}} \mathcal{A}(m, \mathcal{D}).$$

Notice that, in the above definition, $\boldsymbol{v}$ is an n-tuple of constants of Γ_V, where n is the arity of Φ, and $X[\boldsymbol{x}/\boldsymbol{v}]$ denotes the ground atom obtained from $X(\boldsymbol{x})$ by substituting the n-tuple of variables $\boldsymbol{x}$ with $\boldsymbol{v}$. Also, $\mathcal{A}(\mathcal{M}, \mathcal{D})$ is an ABox over the constants $\Gamma = \Gamma_V \cup \tau(\Lambda, \Gamma)$.

Example 7.8. Let $split(\mathcal{OM}_{pr})$ be the *DL-Lite*$_{\mathcal{A},id}$ ontology with split mappings of Example 7.5. Consider in particular the mappings m^a_{21} and m^a_{22} and suppose we have $D_2 = \{(\texttt{20903}, \text{"}\texttt{Rossi}\text{"}), (\texttt{55577}, \text{"}\texttt{White}\text{"})\}$ in the database $\mathcal{D}$. Then, the sets of assertions $\mathcal{A}(m^a_{21}, \mathcal{D})$ and $\mathcal{A}(m^a_{22}, \mathcal{D})$ are as follows:

$$\begin{aligned}
\mathcal{A}(m^a_{21}, \mathcal{D}) &= \{\, \mathit{Employee}(\mathbf{pers}(\texttt{20903})),\ \mathit{Employee}(\mathbf{pers}(\texttt{55577}))\,\} \\
\mathcal{A}(m^a_{22}, \mathcal{D}) &= \{\, \mathbf{persName}(\mathbf{pers}(\texttt{20903}), \text{"}\texttt{Rossi}\text{"}), \\
&\qquad \mathbf{persName}(\mathbf{pers}(\texttt{55577}), \text{"}\texttt{White}\text{"})\,\}
\end{aligned}$$

By proceeding in the same way for each mapping assertion in $split(\mathcal{M}_{pr})$, we easily obtain the whole virtual ABox $\mathcal{A}(\mathcal{M}_{pr}, \mathcal{D}_{pr})$ for $split(\mathcal{OM}_{pr})$, and hence for $\mathcal{OM}_{pr}$. ∎

The following result, which follows easily from the definitions, establishes the relationship between the semantics of *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings and the semantics of *DL-Lite*$_{\mathcal{A},id}$ ontologies by resorting to virtual ABoxes:

Proposition 7.9. *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings. Then we have that*

$$Mod(\mathcal{OM}) = Mod(\langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D})\rangle).$$

Notice that, for convenience, we have defined $\mathcal{A}(\mathcal{M},\mathcal{D})$ for the case where the mappings in $\mathcal{M}$ are split. However, from Proposition 7.6, we also obtain that, for an ontology $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ with non-split mapping assertions, we have that

$$Mod(\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle) = Mod(\langle \mathcal{T}, split(\mathcal{M}), \mathcal{D} \rangle) = Mod(\langle \mathcal{T}, \mathcal{A}(split(\mathcal{M}), \mathcal{D}) \rangle).$$

7.4 Approaches for Query Answering over Ontologies with Mappings

We discuss now in more detail both the bottom-up and the top-down approach for query answering. Proposition 7.9 above suggests an obvious, and "naive", *bottom-up algorithm* to answer queries over a satisfiable *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ with mappings:

1. Materialize the virtual ABox for $\mathcal{OM}$, i.e., compute $\mathcal{A}(\mathcal{M},\mathcal{D})$.
2. Apply to the *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M},\mathcal{D}) \rangle$, the query answering algorithm described in Section 5.

Unfortunately, this approach has the drawback that the resulting algorithm is not anymore AC^0 (or even LOGSPACE) in the size of the database, since it requires the generation and storage of the whole virtual ABox, which in general is polynomial in the size of the database. Moreover, since the database is independent of the ontology, it may happen that, during the lifetime of the ontology with mappings, the data it contains are modified. This would clearly require to set up a mechanism for keeping the virtual ABox up-to-date with respect to the database evolution, similarly to what happens in data warehousing. This is the reason why such a bottom-up approach is only of theoretical interest, but not efficiently realizable in practice.

Hence, we propose a different approach, called "top-down", which uses an algorithm that avoids materializing the virtual ABox, but, rather, takes into account the mapping specification *on-the-fly*, during reasoning. In this way, we can both keep the computational complexity of the algorithm low, which turns out to be as the one of the query answering algorithm for ontologies without mappings (i.e., in AC^0), and avoid any further procedure for data refreshment. We present an overview of our top-down approach to query answering.

Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be a *DL-Lite*$_{\mathcal{A},id}$ ontology with split mappings, and let q be a UCQ [16] over $\mathcal{OM}$. According to the top-down approach, the certain answers to q over $\mathcal{OM}$ are computed by performing the following steps:

1. **Reformulation.** In this step, we compute the perfect reformulation $q_1 = \mathsf{PerfectRefIdC}(q, \mathcal{T})$ of q, according to the technique presented in Section 5. The query q_1 is a UCQ satisfying the following property: the certain answers to q with

[16] Notice that, although we do not consider query answering for UCQs with inequalities, in general we would need to consider also the case where q may contain inequalities. Indeed, such inequalities may result from the queries that encode the violation of a functionality or an identification assertion, and whose evaluation is required to check the satisfiability of $\mathcal{OM}$, cf. Section 5.6. For simplicity we do not consider inequalities here, but they can be dealt with by replacing them with a suitable predicate, which in the end gets translated into an SQL inequality check, see [75] for more details.

respect to $\mathcal{OM}$ coincide with the set of tuples computed by evaluating q_1 over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$[17], i.e., the database representing $\mathcal{A}(\mathcal{M}, \mathcal{D})$.

2. **Filtering.** In this step we take care of a particular problem that the CQs in q_1 might have. Specifically, such a CQ is called *ill-typed* if it has at least one join variable x appearing in two incompatible positions in the query, i.e., such that the TBox $\mathcal{T}$ of the ontology logically implies that x is both of type T_i, and of type T_j, with $i \neq j$ (we remind that in *DL-Lite*$_{\mathcal{A},id}$, data types are pairwise disjoint). The purpose of the filtering step is to remove from the query q_1 all the ill-typed CQs. Intuitively, such a step is needed because the query q_1 has to be unfolded and then evaluated over the source database $\mathcal{D}$ (cf. the next two steps of the algorithm). These last two steps, performed for an ill-typed CQ might produce incorrect results. Let q_2 be the UCQ produced as result of this step.
3. **Unfolding.** Instead of materializing $\mathcal{A}(\mathcal{M}, \mathcal{D})$ and evaluating q_2 over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$ (as in the bottom-up approach), we "unfold" q_2 according to $\mathcal{M}$, i.e., we compute a new query q_3, which is an SQL query over the source relations. As shown in detail in [75], and illustrated briefly below, this computation is done by using logic programming techniques. It allows us to get rid of $\mathcal{M}$, in the sense that the set of tuples computed by evaluating the SQL query q_3 over the database $\mathcal{D}$ coincides with the set of tuples computed by evaluating q_2 over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$.
4. **Evaluation.** The evaluation step consists simply in delegating the evaluation of the SQL query q_3, produced by the unfolding step, over the database $\mathcal{D}$ to the DBMS managing such a database. Formally, such an evaluation returns $ans(q_3, \mathcal{D})$, i.e., the set of tuples obtained from the evaluation of q_3 over $\mathcal{D}$.

The unfolding step for q_2 can be carried out as follows:

(3a) We introduce for each non-split mapping assertion $m_i = \Phi_i(\boldsymbol{x}) \rightsquigarrow \Psi_i(\boldsymbol{y}, \boldsymbol{t})$ in $\mathcal{M}$ an auxiliary predicate Aux_i of the same arity as Φ_i. Intuitively, Aux_i denotes the result of the evaluation over $\mathcal{D}$ of the SQL query Φ_i in the left-hand side of the mapping.

(3b) We introduce for each atom $X(\boldsymbol{y}, \boldsymbol{t})$ in $\Psi_i(\boldsymbol{y}, \boldsymbol{t})$, a logic programming clause

$$X(\boldsymbol{y}, \boldsymbol{t}) \leftarrow Aux_i(\boldsymbol{x}).$$

Notice that, in general, the atom $X(\boldsymbol{y}, \boldsymbol{t})$ in the mapping will contain not only variables but also variable terms, and hence such a clause will contain function symbols in its head.

(3c) From each CQ q' in q_2, we obtain a set of CQs expressed over the Aux_i predicates by *(i)* finding, in all possible ways, the most general unifier ϑ between all atoms in q' and the heads $X(\boldsymbol{y}, \boldsymbol{t})$ of the clauses introduced in the previous step, *(ii)* replacing in q' each head of a clause with the corresponding body, and *(iii)* applying to the resulting CQ the most general unifier ϑ.

[17] The function $DB(\cdot)$ is defined in Section 2.6.

(3d) From the resulting UCQ over the Aux_i predicates, we obtain an SQL query that is a union of select-project-join queries, by substituting each Aux_i predicate with the corresponding SQL query Φ_i.

We refer to [75] for more details, and illustrate the steps above by means of an example.

Example 7.10. Consider the ontology $\mathcal{OM}_{pr}$ of Example 7.2, and assume it is satisfiable. The mapping assertions in $\mathcal{M}_{pr}$ of $\mathcal{OM}_{pr}$ can be encoded in the following portion of a logic program, where for each mapping assertion m_i^a in Figure 22, we have introduced an auxiliary predicate Aux_i:

$$\begin{array}{rcl}
\mathit{TempEmp}(\mathbf{pers}(s)) & \leftarrow & Aux_1(s,p,d) \\
\mathit{WORKS\text{-}FOR}(\mathbf{pers}(s),\mathbf{proj}(p)) & \leftarrow & Aux_1(s,p,d) \\
\mathbf{projName}(\mathbf{proj}(p),p) & \leftarrow & Aux_1(s,p,d) \\
\mathbf{until}(\mathbf{pers}(s),d) & \leftarrow & Aux_1(s,p,d) \\
\mathit{Employee}(\mathbf{pers}(s)) & \leftarrow & Aux_2(s,n) \\
\mathbf{persName}(\mathbf{pers}(s),n) & \leftarrow & Aux_2(s,n) \\
\mathit{Manager}(\mathbf{pers}(s)) & \leftarrow & Aux_3(s,n) \\
\mathbf{persName}(\mathbf{pers}(s),n) & \leftarrow & Aux_3(s,n) \\
\mathit{Manager}(\mathbf{mgr}(c)) & \leftarrow & Aux_4(c,n) \\
\mathbf{persName}(\mathbf{mgr}(c),n) & \leftarrow & Aux_4(c,n)
\end{array}$$

Now, consider the query over $\mathcal{OM}$

$$q(x,n) \;\leftarrow\; \mathit{WORKS\text{-}FOR}(x,y),\ \mathbf{persName}(x,n).$$

Its reformulation $q_1 = \mathsf{PerfectRef}(q,\mathcal{T})$, computed according to the technique presented in Section 5.2, is the UCQ

$$\begin{array}{rcl}
q_1(x,n) & \leftarrow & \mathit{WORKS\text{-}FOR}(x,y),\ \mathbf{persName}(x,n) \\
q_1(x,n) & \leftarrow & \mathbf{until}(x,y),\ \mathbf{persName}(x,n) \\
q_1(x,n) & \leftarrow & \mathit{TempEmp}(x),\ \mathbf{persName}(x,n) \\
q_1(x,n) & \leftarrow & \mathit{Employee}(x),\ \mathbf{persName}(x,n) \\
q_1(x,n) & \leftarrow & \mathit{Manager}(x),\ \mathbf{persName}(x,n)
\end{array}$$

One can verify that in this case none of the CQs of q_1 will be removed by the filtering step, hence $q_2 = q_1$. In order to compute the unfolding of q_2, we unify each of its atoms in all possible ways with the left-hand side of the mapping assertions in $split(\mathcal{M}_{pr})$, i.e., with the heads of the clauses introduced in Step (3b) above, and we obtain the following UCQ q_2':

$$\begin{array}{rcl}
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_1(s,p,d),\ Aux_2(s,n) \\
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_1(s,p,d),\ Aux_3(s,n) \\
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_2(s,n),\ Aux_2(s,n) \\
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_2(s,n),\ Aux_3(s,n) \\
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_3(s,n),\ Aux_2(s,n) \\
q_2'(\mathbf{pers}(s),n) & \leftarrow & Aux_3(s,n),\ Aux_3(s,n) \\
q_2'(\mathbf{mgr}(c),n) & \leftarrow & Aux_4(c,n),\ Aux_4(c,n)
\end{array}$$

```
SELECT CONCAT(CONCAT('pers (',D1.SSN),')'), D2.NAME
FROM D1, D2
WHERE D1.SSN = D2.SSN
  UNION
SELECT CONCAT(CONCAT('pers (',D1.SSN),')'), D3.NAME
FROM D1, D3, D4
WHERE D1.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D2.NAME
FROM D2
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D3.NAME
FROM D2, D3, D4
WHERE D2.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D3.NAME
FROM D2, D3, D4
WHERE D2.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D3.SSN),')'), D3.NAME
FROM D3, D4
WHERE D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('mgr (',D3.CODE),')'), D3.NAME
FROM D3
WHERE D3.CODE NOT IN (SELECT D4.CODE FROM D4)
```

Fig. 24. The SQL query for the projects example produced by the query answering algorithm

Notice that each of the clauses in q_2' is actually generated in many different ways from q_1 and the clauses above.

From q_2', it is now possible to derive the SQL query q_3 shown in Figure 24, where in the derivation we have assumed that duplicate atoms in a clause are eliminated. The SQL query q_3 can be directly issued over the database $\mathcal{D}$ to produce the requested certain answers. □

It is also possible to show that the above procedure can also be used to check satisfiability of an ontology with mappings, by computing the answer to the boolean query that encodes the violation of the constraints in the TBox, and checking whether such an answer is empty.

Let the algorithms for satisfiability and query answering over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings resulting from the above described method be called $\mathsf{SatisfiableDB}(\mathcal{OM})$ and $\mathsf{AnswerDB}(q,\mathcal{OM})$, respectively. A complexity analysis of the various steps of these algorithms, allows us to establish the following result, for whose proof we refer to [75].

Theorem 7.11. *Given a DL-Lite*$_{\mathcal{A},id}$ *ontology with mappings* $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D}\rangle$, *and a UCQ q over $\mathcal{OM}$, both* $\mathsf{SatisfiableDB}(\mathcal{OM})$ *and* $\mathsf{AnswerDB}(q,\mathcal{OM})$ *run in* AC^0 *in the size of $\mathcal{D}$ (data complexity), in polynomial time in the size of $\mathcal{M}$, and in polynomial time in the size of $\mathcal{T}$. Moreover,* $\mathsf{AnswerDB}(q,\mathcal{OM})$ *runs in exponential time in the size of Q.*

7.5 Extending the Mapping Formalism

We investigate now the impact of extending the language used to express the mapping on the computational complexity of query answering. In particular, we consider so-called GLAV mappings [63], i.e., assertions that relate CQs over the database to CQs over the ontology. Such assertions are therefore an extension of both the GAV mappings considered above, and of LAV mappings typical of the data integration setting. Unfortunately, even with LAV mappings only, i.e., mappings where the query over the database simply returns the instances of a single relation, instance checking and query answering are no more in AC^0 with respect to data complexity [20].

Theorem 7.12. *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be an ontology with mappings, where the mapping $\mathcal{M}$ is constituted by a set of LAV mapping assertions. Instance checking (and hence CQ and UCQ query answering) over $\mathcal{OM}$ is* NLOGSPACE*-hard in the size of $\mathcal{D}$.*

Proof. The proof is again by a LOGSPACE reduction from reachability in directed graphs. Let $G = \langle V, E \rangle$ be a directed graph, where V is a set of vertexes and E a set of directed edges, and let s and t be two vertexes of G. As in the proof of Theorem 6.6, we consider the graph represented through first-child and next-sibling functional relations F, N, S (cf. Figure 17).

We define the ontology with LAV mappings $\mathcal{OM}_{lav} = \langle \mathcal{T}_{lav}, \mathcal{M}_{lav}, \mathcal{D}_{lav} \rangle$ as follows:

– The alphabet of $\mathcal{T}_{lav}$ consists of the atomic concepts A and A' and of the atomic roles P_F, P_N, P_S, P_0, and P_{copy}. The TBox itself imposes only that all roles are functional, i.e.,

$$\mathcal{T}_{lav} = \{(\mathsf{funct}\ P_0), (\mathsf{funct}\ P_{copy})\} \cup \{(\mathsf{funct}\ P_{\mathcal{R}}) \mid \mathcal{R} \in \{F, N, S\}\}.$$

– The schema of $\mathcal{D}$ contains a unary relational table A_d and three binary relational tables F_d, N_d, and S_d.
– The LAV mapping $\mathcal{M}_{lav}$ is defined as follows:[18]

$$\begin{array}{rcl} A_d(x) & \rightsquigarrow & q_A(x) \leftarrow A(x), P_{copy}(x,x'), P_0(z,x), P_0(z,x') \\ \mathcal{R}_d(x,y) & \rightsquigarrow & q_{\mathcal{R}}(x,y) \leftarrow P_{\mathcal{R}}(x,y), P_{copy}(x,x'), \\ & & \qquad P_{\mathcal{R}}(x',y'), P_{copy}(y,y'), A'(y'), \quad \text{for } \mathcal{R} \in \{F,N,S\}. \end{array}$$

Figure 25 provides a graphical representation of the kinds of interpretations generated by the LAV mapping above. Notice that the TBox $\mathcal{T}_{lav}$ and the mapping $\mathcal{M}_{lav}$ do not depend on the graph G.

Then, from the graph G and the two vertexes s, t, we define the instance D_G of the database $\mathcal{D}_{lav}$ as follows:

$$D_G = \{\mathcal{R}_d(a,b) \mid (a,b) \in \mathcal{R},\ \text{for } \mathcal{R} \in \{F,N,S\}\} \cup \{A_d(s)\}$$

[18] For simplicity, we do not include function symbols in the mapping since they would play no role in the reduction. Also, instead of using SQL code, we denote the query over the database $\mathcal{D}$ simply by the relation that is returned.

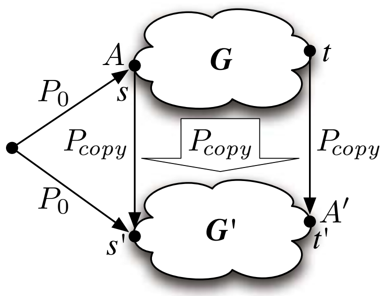


Fig. 25. Interpretation generated by the LAV mapping used in the proof of Theorem 7.12

Intuitively, D_G is simply constituted by the binary relations F_d, N_d, and S_d, used to represent the graph G, and a unary relation A_d containing only s.

Now consider the concept A'. It is possible to show by induction on the length of a path from s to t in G that t is reachable from s in G if and only if $\mathcal{OM}_{lav} \models A'(t)$, i.e., t is an instance of A' in every model of $\mathcal{OM}_{lav}$. Intuitively, this is due to the fact that the functionality of the roles of $\mathcal{T}_{lav}$ forces the objects corresponding to the nodes of G and retrieved through the mapping to be unified with their "copies" generated by the existentially quantified variables in the mapping. Hence, the node t will be forced to become an instance of A' if it is connected to s, but not otherwise. $\square$

The above result shows that, if we allowed more general forms of mappings than the ones considered here for ontologies with mappings, such as LAV mappings, we would lose FOL-rewritability of inference.

Notice that for the above proof to go through, the presence of functionality assertions is crucial. Indeed, it is possible to show that, without functionality assertions (and without identification assertions), query answering even in the presence of GLAV mappings can be done in AC^0 in data complexity, essentially by transforming the GLAV mapping into GAV mappings and introducing additional constraints (and relations of arbitrary arity) in the TBox [16].

8 Ontology-Based Data Access Software Tools

In this section we introduce two tools specifically designed for OBDA as described in the previous sections, namely DIG-QuOnto and the OBDA Plugin for Protégé 3.3.1. The first, presented in Section 8.1, is a server for the QuOnto reasoner [2,76] that exposes it's reasoning services and OBDA functionality through an extended version of the DIG Interface [10], a standard communication protocol for DL reasoners. The second, presented in Section 8.2, is a plugin for the ontology editor Protégé[19] that provides facilities to model *ontologies with mappings* (see Section 7), to *synchronize* these models with an OBDA enabled reasoner through an extended DIG protocol, and to access CQ services offered by DIG 1.2 compatible reasoners [76].

[19] `http://protege.stanford.edu/`

Both tools can be used together in order to design, deploy and use a fully functional OBDA layer on top of existing relational databases.

8.1 DIG-QuOnto, the OBDA-DIG Server for QuOnto

DIG-QuOnto [76] is a module for the QuOnto system that exposes the functionality of the QuOnto reasoner and its *RDBMS-ontology mapping* module through an extended version of the DIG 1.1 Interface [10], the HTTP/XML based communication protocol for DL reasoners. Using DIG-QuOnto, it is possible to extend an existing relational database with an OBDA layer, which can be used to cope with incomplete information or function as a data integration layer.

The QuOnto reasoner is a DL reasoner that implements the reasoning and query answering algorithms for *DL-Lite*$_{\mathcal{A},id}$ presented in Sections 4 and 5. Built on top of the QuOnto reasoner is its *RDBMS-ontology mapping* module, a module that implements the mapping techniques described in Section 7. DIG-QuOnto wraps both components, thus combining their functionalities in a common interface accessible to clients and providing several features that we will now describe.

Ontology Representation. QuOnto and DIG-QuOnto work with *ontologies with mappings* of the form $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ as presented in Section 7, where $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A},id}$ TBox, $\mathcal{D}$ is a relational database (constituted by the relational schema, and the extensions of the relations), and $\mathcal{M}$ is a set of mapping assertions. As mentioned, $\mathcal{D}$ and $\mathcal{M}$ together define an ABox $\mathcal{A}(\mathcal{M}, \mathcal{D})$, which however is never materialized by QuOnto. Instead, using the *RDBMS-ontology mapping* module, QuOnto is able to rewrite an UCQ q over $\mathcal{T}$ into an SQL query that is executed by the RDBMS managing $\mathcal{D}$ and that retrieves the answers to q. Hence, QuOnto is able to exploit many optimizations available in modern RDBMS engines in all operations related to the extensional level of the ontology, and as a consequence, it is able to handle large amounts of data. A further consequence is that data has never to be imported into the ontology, as is done, e.g., with OWL ontologies. Instead, in DIG-QuOnto it is possible to let the data reside in the original relational source. Note that this has the consequence that if the data source is not a traditional database, but in fact a virtual schema created by a data federation tool, DIG-QuOnto would act as a highly efficient data integration system.

For a detailed description of the reasoning process used in DIG-QuOnto see Sections 4, 5 and 7.

Reasoning Services. QuOnto provides the following reasoning services over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$:

Answering UCQs. Given $\mathcal{OM}$ and a UCQ q, compute the certain answers for q over $\mathcal{OM}$. This is the central reasoning service of the QuOnto system, to which all other reasoning services are reduced. This service is unique in QuOnto in that, at the moment of writing, QuOnto is the only DL reasoner that offers UCQ query answering under the *standard certain answer* semantics (as opposed to weaker semantics, such as the *grounded semantics*, implemented in other reasoners). This is especially important in settings of incomplete information (e.g., the data integration setting), as it allows QuOnto to bind variables in the body of queries to *unknown* individuals which are only deduced to exist due to $\mathcal{T}$ but which are not produced by the data in $\mathcal{D}$ (considering the mappings $\mathcal{M}$).

The implementation of this service is based on the algorithms described in Sections 4, 5, and 7.

Checking ontology satisfiability. Given $\mathcal{OM}$, check if $\mathcal{OM}$ has a model. This is a classical reasoning service available in most DL reasoners, which mixes intensional and extensional reasoning. In QuOnto, this service is reduced to query answering. Specifically, to check satisfiability of $\mathcal{OM}$, QuOnto checks whether the answer to the query that checks for a violation of one of the negative inclusion, functionality, or identification assertions in the TBox is empty, as illustrated in Section 5.6. Hence, it can be done efficiently in the size of the data, which is maintained in the database and managed through a DBMS.

Checking atomic satisfiability. Given $\mathcal{OM}$ and an atomic entity X (i.e., a concept, a role, or an attribute) in the alphabet of $\mathcal{OM}$, check whether there exists a model $\mathcal{I}$ of $\mathcal{OM}$ such that $X^{\mathcal{I}} \neq \emptyset$. This is a purely intensional reasoning service, i.e., it actually depends only on the TBox $\mathcal{T}$ of $\mathcal{OM}$. In QuOnto, this service is reduced to checking ontology satisfiability, as illustrated in Section 4.4, and hence ultimately to query answering. In fact, since satisfiability of the entity X depends only on the TBox of $\mathcal{OM}$ (and not on the database and mappings), QuOnto constructs for the purpose an ad-hoc database and mappings, that are distinct from the ones of $\mathcal{OM}$, and answers the appropriate queries over such a database.

Checking subsumption. Given $\mathcal{OM}$ and two atomic entities X_1 and X_2 in the alphabet of $\mathcal{OM}$, check whether $X_1^{\mathcal{I}} \subseteq X_2^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{OM}$. In QuOnto, also subsumption is reduced to ontology satisfiability, as illustrated in Section 4.4.

Queries regarding the concept/role hierarchy. Given $\mathcal{OM}$, QuOnto provides the following intensional reasoning services regarding the structure of the concept and role hierarchy in $\mathcal{T}$:

- Ancestors. Given an atomic entity X in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities X' such that $\mathcal{T} \models X \sqsubseteq X'$.
- Parents. Given an atomic entity X in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities X' such that $\mathcal{T} \models X \sqsubseteq X'$ and there is no X'' such that $\mathcal{T} \models X \sqsubseteq X''$ and $\mathcal{T} \models X'' \sqsubseteq X'$. This corresponds to the immediate subsumption relation.
- Descendants. Given an atomic entity X in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities X' such that $\mathcal{T} \models X' \sqsubseteq X$.
- Children. Given an atomic entity X in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities X' such that $\mathcal{T} \models X' \sqsubseteq X$ and there is no X'' such that $\mathcal{T} \models X' \sqsubseteq X''$ and $\mathcal{T} \models X'' \sqsubseteq X$.
- Equivalents. Given an atomic entity X in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities X such that $\mathcal{T} \models X \sqsubseteq X'$ and $\mathcal{T} \models X' \sqsubseteq X$.

In QuOnto, such services are also ultimately reduced to query answering, although for efficiency reasons this is done in an ad-hoc manner.

Checking implication of assertions. Given $\mathcal{OM}$ and a functionality or identification assertion α, check whether $\mathcal{T} \models \alpha$. We point out that implication of identification assertions is unique to QuOnto, given the fact that these kinds of constraints are not available in most DL reasoners.

The OBDA-DIG Communication Layer. All functionalities of DIG-QuOnto are accessible through an extended version of the DIG 1.1 Interface [10], which is an effort carried out by the DL Implementation Group (DIG) to standardize interaction with DL reasoners in a networked environment. The original specification defines the communication mechanism to which DL reasoners (i.e., DIG servers) and clients comply. The interface has been widely accepted and most well known ontology design tools and DL reasoners implement it. XML messages, divided in `tells` and `asks`, allow the client to: *(i)* query for the server's reasoning capabilities, *(ii)* transfer the assertions of an ontology to the server, *(iii)* perform certain ontology manipulations, and *(iv)* ask standard DL queries about the given ontology, e.g., concept subsumption, satisfiability, equivalence, etc. The concept language used to describe DIG ontologies is based on the $\mathcal{SHOIQ}(D)$ description logic [53].

As mentioned above, DIG-QuOnto offers ontology constructs and services not available in traditional DL reasoners and therefore not considered in DIG 1.1. Hence, it has been necessary to implement not only the original DIG 1.1 interface, but also extensions that enable the use of the functionality available in DIG-QuOnto.

Being QuOnto's main reasoning task answering UCQs, the DIG-QuOnto server implements the so called DIG 1.2 specification [77], an extension to the original DIG 1.1 interface that provides an ABox query language for DIG clients and server. Concretely, it provides the ability to pose UCQs. It doesn't restrict the semantics for the expressed queries, hence in DIG-QuOnto we use *traditional certain answer semantics*, as described in Section 2.4. At the moment of writing, DIG 1.2 is implemented in the DIG modules for the RacerPro[20] and QuOnto reasoners.

The core component of the protocol implemented in the DIG-QuOnto server are the OBDA extensions to DIG 1.1 [80,81]. These extensions have as main objective to augment DIG 1.1 with the concepts of *Data Source* and *Mapping*, which are at the core of the OBDA setting and fundamental to the functionality offered by the *rdbms-ontology mapping* module for QuOnto. Moreover, the extension aims at the standardization of the interaction with reasoners offering OBDA functionality, not only with QuOnto. For more information about the OBDA extension to DIG 1.1 we refer to the extension's website[21].

DIG-QuOnto can be executed as a user service or as a system wide service. Once initiated, DIG-QuOnto listens for DIG-OBDA requests issued by clients. A number of parameters are available at initialization time. These allow the user to indicate whether DIG-QuOnto should perform automatic consistency checking at query time, and whether it should use *view based* unfolding procedures, among other things.

[20] `http://www.racer-systems.com/`

[21] `http://obda.inf.unibz.it/dig-11-obda/`

The status and operation of DIG-QuOnto can be monitored with its web interface, which is available at:

```
http://[QUONTOHOST]:[QUONTOPORT]/index.jsp
```

Through the interface, users can obtain information such as the ontologies currently loaded into the system, review system logs, or visualize system parameters.

Regarding implementation details, we note that DIG-QuOnto is written in Java and requires Sun's Java Runtime Environment (JRE)[22]. Moreover, DIG-QuOnto uses Java JDBC connectors to establish communication with a DBMS. The following DBMSs are supported by DIG-QuOnto at the moment of writing: MySQL 5.0.45[23], PostgreSQL 8.3, Oracle 10g and 11g , DB2 8.2 and 9.1, SQLite 3.5.9, H2 1.0.74, and Derby 10.3.2.1. We refer to QuOnto's main website[24] for detailed information about the software and for download links.

8.2 The OBDA Plugin for Protégé

The OBDA Plugin [81,82] is an open source add-on for the ontology editor Protégé 3.3.1 (see Footnote 19) whose main objectives are, on the one hand, to extend Protégé so as to allow its use in the design of *ontologies with mappings* and, on the other hand, to extend the way in which Protégé interacts with DL reasoners so as to support the interaction with reasoners that are designed for the OBDA architecture and that can make use of the OBDA information introduced using the plugin's facilities. These objectives are accomplished by extending the GUI, back-end mechanisms, and communication protocols of Protégé. In the following paragraphs, we will briefly describe these extensions, referring to the plugin's website[25] for a comprehensive overview of all the features of the OBDA Plugin.

Database and Mapping Definition. The main GUI component of the OBDA Plugin is the *Datasource Manager* tab (see Figure 26). Using this tab, users are able to associate JDBC data sources to the currently open ontology. Moreover, for each data source, users are able to define a set of mappings that relate the data returned by an arbitrary SQL query over the source to the entities (i.e., classes and properties) of the ontology.

The plugin offers several features to facilitate these tasks. Among them we can find simple ones, such as syntax coloring for the mapping editor or database connection validation, and more complex ones, such as SQL query validation, direct DBMS query facilities (see Figure 27), and database schema inspection facilities (see Figure 28).

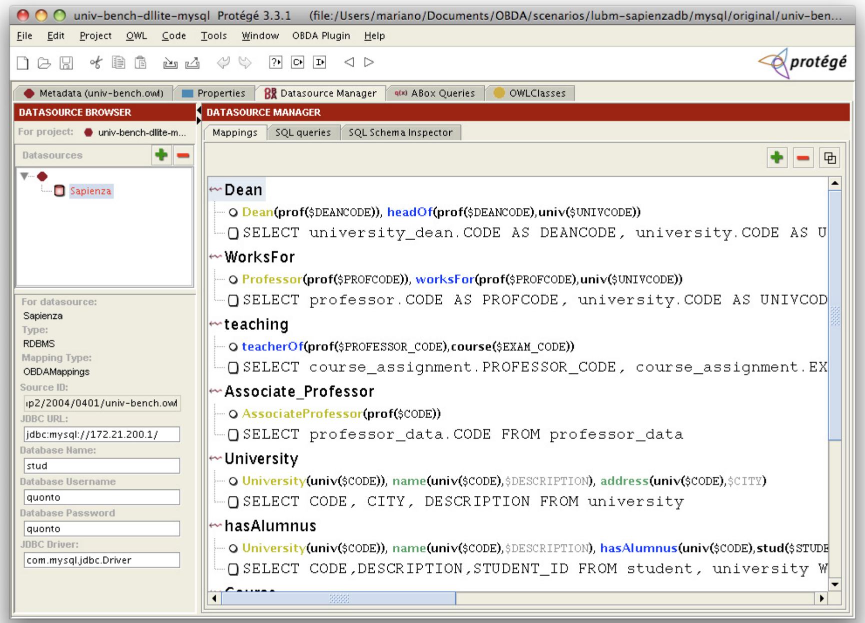
All data introduced with the plugin's GUI components is made persistent in so called `.obda` files. These are XML files that can be read and edited easily by users or applications and which are independent from Protégé's `.owl` and `.ppjr` files. This feature enables users to easily extend existing OWL ontologies with OBDA features without compromising the original model.

[22] `http://java.sun.com/javase/`

[23] The use of MySQL is highly discouraged due to performance limitations of this engine.

[24] `http://www.dis.uniroma1.it/~quonto/`

[25] `http://obda.inf.unibz.it/protege-plugin/`


univ-bench-dllite-mysql Protégé 3.3.1 (file:/Users/mariano/Documents/OBDA/scenarios/lubm-sapienzadb/mysql/original/univ-ben...
File Edit Project OWL Code Tools Window OBDA Plugin Help
protégé
Metadata (univ-bench.owl)
Properties
Datasource Manager
ABox Queries
OWLClasses
DATASOURCE BROWSER
DATASOURCE MANAGER
Mappings
SQL queries
SQL Schema Inspector
Datasources

For project: univ-bench-dllite-m...

Sapienza

For datasource:
Sapienza
Type:
RDBMS
Mapping Type:
OBDAMappings
Source ID:
ıp2/2004/0401/univ-bench.owl
JDBC URL:
jdbc:mysql://172.21.200.1/
Database Name:
stud
Database Username
quonto
Database Password
quonto
JDBC Driver:
com.mysql.jdbc.Driver

```
Dean
  Dean(prof($DEANCODE)), headOf(prof($DEANCODE),univ($UNIVCODE))
  SELECT university_dean.CODE AS DEANCODE, university.CODE AS U
WorksFor
  Professor(prof($PROFCODE)), worksFor(prof($PROFCODE),univ($UNIVCODE))
  SELECT professor.CODE AS PROFCODE, university.CODE AS UNIVCOD
teaching
  teacherOf(prof($PROFESSOR_CODE),course($EXAM_CODE))
  SELECT course_assignment.PROFESSOR_CODE, course_assignment.EX
Associate_Professor
  AssociateProfessor(prof($CODE))
  SELECT professor_data.CODE FROM professor_data
University
  University(univ($CODE)), name(univ($CODE),$DESCRIPTION), address(univ($CODE),$CITY)
  SELECT CODE, CITY, DESCRIPTION FROM university
hasAlumnus
  University(univ($CODE)), name(univ($CODE),$DESCRIPTION), hasAlumnus(univ($CODE),stud($STUDE
  SELECT CODE,DESCRIPTION,STUDENT_ID FROM student, university W
```

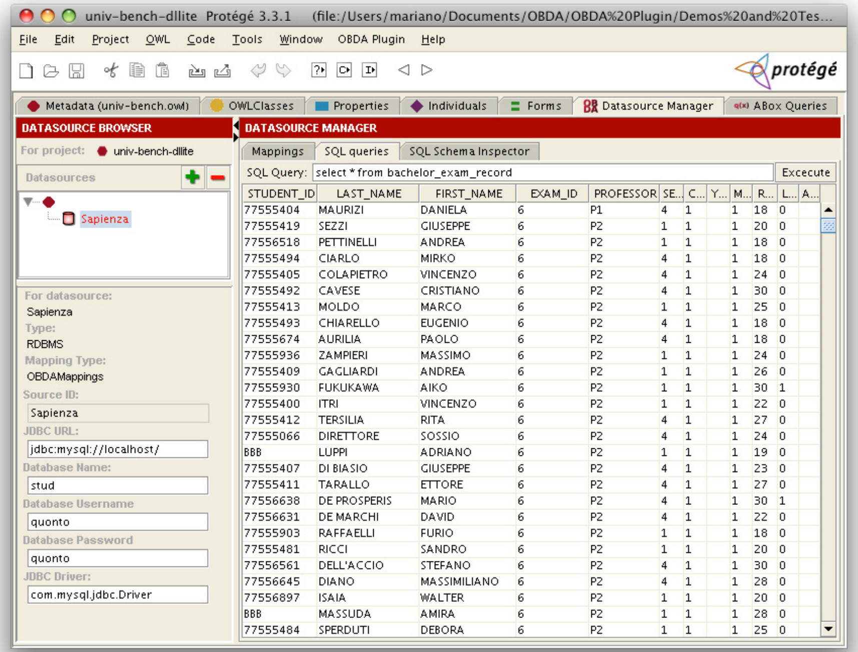
Fig. 26. OBDA Plugin's Datasource Manager tab

OBDA-DIG Synchronization for DL Reasoners. The mechanism used in Protégé 3.3.1 for interaction with DL reasoners is the DIG 1.1 Interface [10]. In order to allow for the interaction with reasoners designed for the OBDA setting, the OBDA Plugin extends Protégé's DIG implementation with the *OBDA Extensions for DIG* [80,81]. Using these extensions, the plugin is able to transmit the OBDA model created by the user to any reasoner implementing both specifications (see Section 8.1 for a short overview of these protocols).

Moreover, the plugin offers several possibilities to tweak the way in which synchronization takes place. For example, to interact with a traditional DL reasoner while the OBDA Plugin is installed, it is possible to configure the OBDA Plugin so as to use the original DIG 1.1 synchronization mechanism instead of the extended one.

UCQs with the OBDA Plugin. Another key feature of the OBDA Plugin is its ability to interact with reasoners that offer the service of answering (U)CQs. Using the *ABox Queries* tab of the OBDA Plugin (see Figure 29), users are able to express UCQs written in a SPARQL-like syntax. To provide the answers to the query, the OBDA Plugin translates the UCQ to a DIG 1.2 [77] request, which is sent to the reasoner. Clearly, the target reasoner must provide a UCQ answering service through a DIG 1.2 interface. At the moment of writing there exist two such systems, namely the DIG-QuOnto server and the Racer++ system.

In addition to the basic query handling facilities, the *ABox Queries* tab offers extra functionality such as persistent support for query *collections*, batch mode result retrieval and result set export facilities.


univ-bench-dllite Protégé 3.3.1 (file:/Users/mariano/Documents/OBDA/OBDA%20Plugin/Demos%20and%20Tes...
File Edit Project OWL Code Tools Window OBDA Plugin Help
protégé
Metadata (univ-bench.owl)
OWLClasses
Properties
Individuals
Forms
Datasource Manager
ABox Queries
DATASOURCE BROWSER
DATASOURCE MANAGER
Mappings
SQL queries
SQL Schema Inspector
Datasources
Sapienza
Excecute

For project: univ-bench-dllite

For datasource: Sapienza

Type: RDBMS

Mapping Type: OBDAMappings

Source ID: Sapienza

JDBC URL: jdbc:mysql://localhost/

Database Name: stud

Database Username: quonto

Database Password: quonto

JDBC Driver: com.mysql.jdbc.Driver

SQL Query: select * from bachelor_exam_record

STUDENT_ID	LAST_NAME	FIRST_NAME	EXAM_ID	PROFESSOR	SE...	C...	Y...	M...	R...	L...	A...
77555404	MAURIZI	DANIELA	6	P1	4	1		1	18	0	
77555419	SEZZI	GIUSEPPE	6	P2	1	1		1	20	0	
77556518	PETTINELLI	ANDREA	6	P2	1	1		1	18	0	
77555494	CIARLO	MIRKO	6	P2	4	1		1	18	0	
77555405	COLAPIETRO	VINCENZO	6	P2	4	1		1	24	0	
77555492	CAVESE	CRISTIANO	6	P2	4	1		1	30	0	
77555413	MOLDO	MARCO	6	P2	1	1		1	25	0	
77555493	CHIARELLO	EUGENIO	6	P2	4	1		1	18	0	
77555674	AURILIA	PAOLO	6	P2	4	1		1	18	0	
77555936	ZAMPIERI	MASSIMO	6	P2	1	1		1	24	0	
77555409	GAGLIARDI	ANDREA	6	P2	1	1		1	26	0	
77555930	FUKUKAWA	AIKO	6	P2	1	1		1	30	1	
77555400	ITRI	VINCENZO	6	P2	1	1		1	22	0	
77555412	TERSILIA	RITA	6	P2	4	1		1	27	0	
77555066	DIRETTORE	SOSSIO	6	P2	4	1		1	24	0	
BBB	LUPPI	ADRIANO	6	P2	1	1		1	19	0	
77555407	DI BIASIO	GIUSEPPE	6	P2	4	1		1	23	0	
77555411	TARALLO	ETTORE	6	P2	4	1		1	27	0	
77556638	DE PROSPERIS	MARIO	6	P2	4	1		1	30	1	
77556631	DE MARCHI	DAVID	6	P2	4	1		1	22	0	
77555903	RAFFAELLI	FURIO	6	P2	1	1		1	18	0	
77555481	RICCI	SANDRO	6	P2	1	1		1	20	0	
77556561	DELL'ACCIO	STEFANO	6	P2	4	1		1	30	0	
77556645	DIANO	MASSIMILIANO	6	P2	4	1		1	28	0	
77556897	ISAIA	WALTER	6	P2	1	1		1	20	0	
BBB	MASSUDA	AMIRA	6	P2	1	1		1	28	0	
77555484	SPERDUTI	DEBORA	6	P2	1	1		1	25	0	

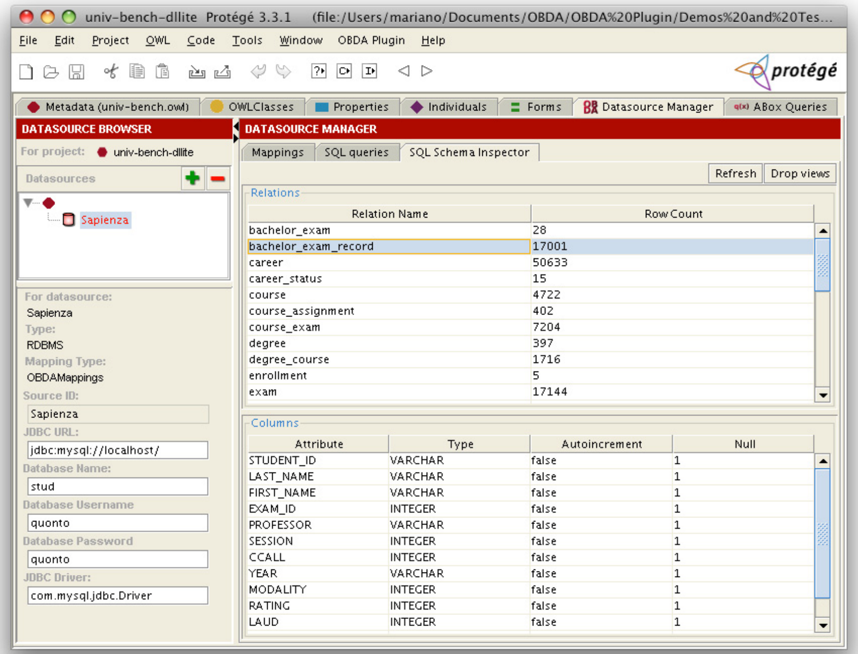
Fig. 27. OBDA Plugin's direct SQL Query tab

DIG-QUONTO Specific Features. Since the OBDA Plugin is being developed in parallel with DIG-QUONTO, the plugin incorporates several DIG-QUONTO specific facilities. These include the ability to enable and disable reasoning during query answering, to retrieve the expansion/rewriting and unfolding of a UCQ, to visualize the QUONTO TBoxes, and to request ontology satisfiability checking.

Extensible OBDA API. An important feature of the OBDA Plugin is its modular architecture. The OBDA Plugin for Protégé 3.3.1 has been built on top of a Java API for OBDA. The API is independent from the final GUI (e.g., Protégé) and, more importantly, independent from particular characterizations of the OBDA architecture. This enables the API to accommodate for different mapping techniques or data source types, having as only requirement that the mapping technique regards a mappings as a pair composed by a query Φ over a data source and a query Ψ over the ontology. Note that here, the term *query* is used in the most general sense of the word, as a query can stand for any arbitrary computation.

We observe that the GUI independence aspect of the core API has been used to provide a port of the OBDA Plugin for the NeOn Toolkit platform[26] and to build the initial prototypes of the Protégé 4 and Protégé 3.4 versions of the plugin. The core API of the OBDA Plugin will soon be released with an open source license.

[26] `http://obda.inf.unibz.it/neon-plugin/`


univ-bench-dllite Protégé 3.3.1 (file:/Users/mariano/Documents/OBDA/OBDA%20Plugin/Demos%20and%20Tes...
File Edit Project OWL Code Tools Window OBDA Plugin Help
protégé
Metadata (univ-bench.owl) OWLClasses Properties Individuals Forms Datasource Manager ABox Queries
DATASOURCE BROWSER
DATASOURCE MANAGER
Datasources
Mappings SQL queries SQL Schema Inspector
Refresh Drop views
Relations
Columns

For project: univ-bench-dllite

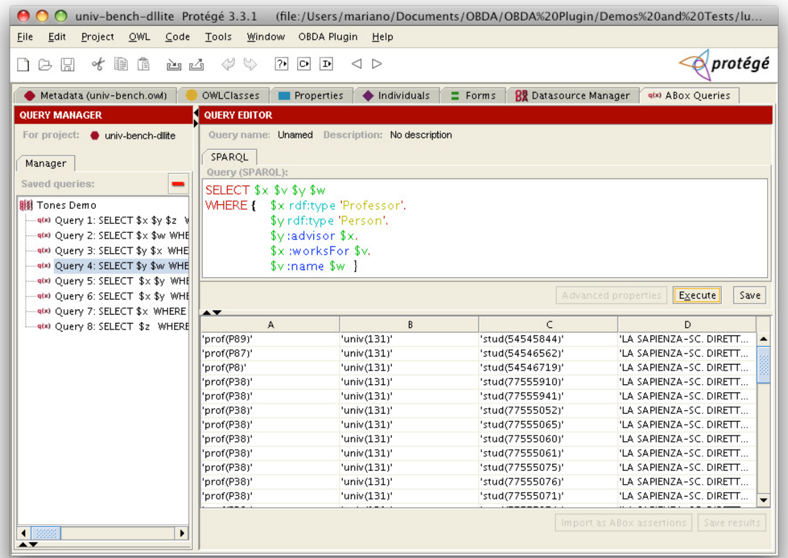
Sapienza

For datasource: Sapienza
Type: RDBMS
Mapping Type: OBDAMappings
Source ID: Sapienza
JDBC URL: jdbc:mysql://localhost/
Database Name: stud
Database Username: quonto
Database Password: quonto
JDBC Driver: com.mysql.jdbc.Driver

Relation Name	Row Count
bachelor_exam	28
bachelor_exam_record	17001
career	50633
career_status	15
course	4722
course_assignment	402
course_exam	7204
degree	397
degree_course	1716
enrollment	5
exam	17144

Attribute	Type	Autoincrement	Null
STUDENT_ID	VARCHAR	false	1
LAST_NAME	VARCHAR	false	1
FIRST_NAME	VARCHAR	false	1
EXAM_ID	INTEGER	false	1
PROFESSOR	VARCHAR	false	1
SESSION	INTEGER	false	1
CCALL	INTEGER	false	1
YEAR	VARCHAR	false	1
MODALITY	INTEGER	false	1
RATING	INTEGER	false	1
LAUD	INTEGER	false	1

Fig. 28. OBDA Plugin's RDBMS Schema Inspector


univ-bench-dllite Protégé 3.3.1 (file:/Users/mariano/Documents/OBDA/OBDA%20Plugin/Demos%20and%20Tests/lu...
File Edit Project OWL Code Tools Window OBDA Plugin Help
protégé
Metadata (univ-bench.owl) OWLClasses Properties Individuals Forms Datasource Manager ABox Queries
QUERY MANAGER
QUERY EDITOR
Manager
SPARQL
Advanced properties Execute Save
Import as ABox assertions Save results

For project: univ-bench-dllite

Query name: Unamed Description: No description

Saved queries:
- Tones Demo
  - Query 1: SELECT $x $y $z W
  - Query 2: SELECT $x $w WHE
  - Query 3: SELECT $y $x WHE
  - Query 4: SELECT $y $w WHE
  - Query 5: SELECT $x $y WHE
  - Query 6: SELECT $x $y WHE
  - Query 7: SELECT $x WHERE
  - Query 8: SELECT $z WHERE

Query (SPARQL):

```
SELECT $x $v $y $w
WHERE {   $x rdf:type 'Professor'.
          $y rdf:type 'Person'.
          $y :advisor $x.
          $x :worksFor $v.
          $v :name $w }
```

A	B	C	D
'prof(P89)'	'univ(131)'	'stud(54545844)'	'LA SAPIENZA-SC. DIRETT...
'prof(P87)'	'univ(131)'	'stud(54546562)'	'LA SAPIENZA-SC. DIRETT...
'prof(P8)'	'univ(131)'	'stud(54546719)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555910)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555941)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555052)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555065)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555060)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555061)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555075)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555076)'	'LA SAPIENZA-SC. DIRETT...
'prof(P38)'	'univ(131)'	'stud(77555071)'	'LA SAPIENZA-SC. DIRETT...

Fig. 29. OBDA Plugin's ABox Queries Tab

9 Conclusions

In this article, we have summarized the main technical results concerning the *DL-Lite* family of description logics, which has been developed, studied, and implemented in recent years as a solution to the problem of ontology-based data access (OBDA). The *DL-Lite* family, and in particular *DL-Lite*$_{\mathcal{A},id}$, an expressive member of this family that we have used as the basis for our technical development, provides a solution to the trade-off between expressiveness of the language and complexity of inference that is optimized towards the requirements arising in OBDA, namely *(i)* the ability to capture the main modeling constructs of conceptual modeling languages, such as UML class diagrams and the Entity-Relationship model, and *(ii)* efficient reasoning over large amounts of data and the ability to compute the certain answers to conjunctive queries and unions thereof w.r.t. an ontology by rewriting a query into a new query and directly evaluating such a rewritten query over the available data using a relational DBMS.

The results we have presented here are based mainly on work that has been carried out in the last years and published in the following articles: [22,24,25,20,75]. However, the *DL-Lite* family has spurred a lot of interest in the research community, and recently various follow-up activities have emerged and research is still very active and ongoing. We summarize here the major research efforts:

– Extensions of the ontology language with further constructs, such as number restrictions (which are a generalization of functionality), full booleans, and additional role constructs present in OWL 2, and a systematic analysis of the computational complexity of inference for all meaningful combinations of constructs and under various assumptions [3,4].
– Extension of the query language to support full first-order queries (or equivalently, SQL queries) under a weakened epistemic semantics, overcoming the undecidability of full first-order inference over ontologies [23].
– Updating *DL-Lite* ontologies [35,36].
– Fuzzy extensions to the DLs of the *DL-Lite* family [87,72].
– Extensions of *DL-Lite* with temporal operators, which has applications to temporal conceptual data modeling [5].
– Computation and extraction of modules from an ontology [60,59].

Acknowledgements. This research has been partially supported by the FET project TONES (Thinking ONtologiES), funded within the EU 6th Framework Programme under contract FP6-7603, and by the large-scale integrating project (IP) OntoRule (ONTOlogies meet Business RULEs), funded by the EC under ICT Call 3 FP7-ICT-2008-3, contract number FP7-231875.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
2. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 1670–1671 (2005)

3. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: DL-Lite in the light of first-order logic. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 361–366 (2007)
4. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. Technical Report BBKCS-09-03, School of Computer Science and Information Systems, Birbeck College, London (2009), `http://www.dcs.bbk.ac.uk/research/techreps/2009/bbkcs-09-03.pdf`
5. Artale, A., Kontchakov, R., Lutz, C., Wolter, F., Zakharyaschev, M.: Temporalising tractable description logics. In: Proc. of the 14th Int. Symp. on Temporal Representation and Reasoning (TIME 2007), pp. 11–22 (2007)
6. Baader, F.: Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence, IJCAI 1991 (1991)
7. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
8. Baader, F., Hladik, J., Lutz, C., Wolter, F.: From tableaux to automata for description logics. Fundamenta Informaticae 57, 1–33 (2003)
9. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69(1), 5–40 (2001)
10. Bechhofer, S., Möller, R., Crowther, P.: The DIG description logic interface. In: Proc. of the 2003 Description Logic Workshop (DL 2003). CEUR Electronic Workshop Proceedings, vol. 81, pp. 196–203 (2003), `http://ceur-ws.org/`
11. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
12. Brachman, R.J., Levesque, H.J.: The tractability of subsumption in frame-based description languages. In: Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI 1984), pp. 34–37 (1984)
13. Brachman, R.J., Levesque, H.J. (eds.): Readings in Knowledge Representation. Morgan Kaufmann, San Francisco (1985)
14. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2), 171–216 (1985)
15. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. J. of Artificial Intelligence Research 1, 109–138 (1993)
16. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 338–350. Springer, Heidelberg (2002)
17. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)
18. Calvanese, D., De Giacomo, G.: Expressive description logics. In: Baader, et al. (eds.) [7], ch. 5, pp. 178–218 (2003)
19. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic $\textit{DL-Lite}_{\mathcal{A}}$. In: Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006). CEUR Electronic Workshop Proceedings, vol. 216 (2006), `http://ceur-ws.org/`
20. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R., Ruzzi, M.: Data integration through $\textit{DL-lite}_{\mathcal{A}}$ ontologies. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 26–47. Springer, Heidelberg (2008)

21. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 602–607 (2005)
22. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
23. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), pp. 274–279 (2007)
24. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
25. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Path-based identification constraints in description logics. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 231–241 (2008)
26. Calvanese, D., De Giacomo, G., Lenzerini, M.: Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 1999), pp. 84–89 (1999)
27. Calvanese, D., De Giacomo, G., Lenzerini, M.: Answering queries using views over description logics knowledge bases. In: Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000), pp. 386–391 (2000)
28. Calvanese, D., De Giacomo, G., Lenzerini, M.: 2ATAs make DLs easy. In: Proc. of the 2002 Description Logic Workshop (DL 2002). CEUR Electronic Workshop Proceedings, vol. 53, pp. 107–118 (2002), `http://ceur-ws.org/`
29. Calvanese, D., De Giacomo, G., Lenzerini, M.: Description logics for information integration. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2408, pp. 41–60. Springer, Heidelberg (2002)
30. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D.: Reasoning in expressive description logics. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 23, vol. II, pp. 1581–1634. Elsevier Science Publishers, Amsterdam (2001)
31. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. J. of Artificial Intelligence Research 11, 199–240 (1999)
32. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th ACM Symp. on Theory of Computing (STOC 1977), pp. 77–90 (1977)
33. Chen, C., Haarslev, V., Wang, J.: LAS: Extending Racer by a Large ABox Store. In: Proc. of the 2005 Description Logic Workshop (DL 2005). CEUR Electronic Workshop Proceedings, vol. 147 (2005), `http://ceur-ws.org/`
34. Cosmadakis, S.S., Kanellakis, P.C., Vardi, M.: Polynomial-time implication problems for unary inclusion dependencies. J. of the ACM 37(1), 15–46 (1990)
35. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), pp. 1271–1276 (2006)
36. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the approximation of instance level update and erasure in description logics. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 403–408 (2007)
37. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology based access to distributed and semi-structured information. In: Meersman, R., Tari, Z., Stevens, S. (eds.) Database Semantic: Semantic Issues in Multimedia Systems, ch. 20, pp. 351–370. Kluwer Academic Publishers, Dordrecht (1999)

38. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: The complexity of concept languages. Information and Computation 134, 1–58 (1997)
39. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Deduction in concept languages: From subsumption to instance checking. J. of Logic and Computation 4(4), 423–452 (1994)
40. Garey, M.R., Johnson, D.S.: Computers and Intractability — A guide to NP-completeness. W. H. Freeman and Company, San Francisco (1979)
41. Goasdoue, F., Lattes, V., Rousset, M.-C.: The use of CARIN language and algorithms for information integration: The Picsel system. Int. J. of Cooperative Information Systems 9(4), 383–401 (2000)
42. Gruber, T.: Towards principles for the design of ontologies used for knowledge sharing. Int. J. of Human and Computer Studies 43(5/6), 907–928 (1995)
43. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition 5(2), 199–220 (1993)
44. Guarino, N.: Formal ontology in information systems. In: Proc. of the Int. Conf. on Formal Ontology in Information Systems (FOIS 1998). Frontiers in Artificial Intelligence, pp. 3–15. IOS Press, Amsterdam (1998)
45. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
46. Halevy, A.Y.: Answering queries using views: A survey. J. of Very Large Database 10(4), 270–294 (2001)
47. Heflin, J., Hendler, J.: A portrait of the Semantic Web in action. IEEE Intelligent Systems 16(2), 54–59 (2001)
48. Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Feier, C., Hench, G., Wetzstein, B., Keller, U.: Ontology reasoning with large data repositories. In: Hepp, M., De Leenheer, P., de Moor, A., Sure, Y. (eds.) Ontology Management, Semantic Web, Semantic Web Services, and Business Applications. Semantic Web And Beyond Computing for Human Experience, vol. 7, pp. 89–128. Springer, Heidelberg (2008)
49. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1998), pp. 636–647 (1998)
50. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL reasoning with large numbers of individuals. In: Proc. of the 2004 Description Logic Workshop (DL 2004). CEUR Electronic Workshop Proceedings, vol. 104 (2004), `http://ceur-ws.org/`
51. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1), 7–26 (2003)
52. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of Logic and Computation 9(3), 385–410 (1999)
53. Horrocks, I., Sattler, U.: A tableau decision procedure for $\mathcal{SHOIQ}$. J. of Automated Reasoning 39(3), 249–276 (2007)
54. Hull, R.: A survey of theoretical research on typed complex database objects. In: Paredaens, J. (ed.) Databases, pp. 193–256. Academic Press, London (1988)
55. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), pp. 466–471 (2005)
56. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. of Computer and System Sciences 28(1), 167–189 (1984)
57. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: Proc. of the 24rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005), pp. 61–75 (2005)

58. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1998), pp. 205–213 (1998)
59. Kontchakov, R., Pulina, L., Sattler, U., Schneider, T., Selmer, P., Wolter, F., Zakharyaschev, M.: Minimal module extraction from DL-Lite ontologies using QBF solvers. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence, IJCAI 2009 (2009)
60. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Can you tell the difference between DL-Lite ontologies? In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 285–295 (2008)
61. Kozen, D.: Theory of Computation. Springer, Heidelberg (2006)
62. Krisnadhi, A., Lutz, C.: Data complexity in the $\mathcal{EL}$ family of description logics. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 333–347. Springer, Heidelberg (2007)
63. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pp. 233–246 (2002)
64. Libkin, L.: Data exchange and incomplete information. In: Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006), pp. 60–69 (2006)
65. Maedche, A.: Ontology learning for the Semantic Web. Kluwer Academic Publishers, Dordrecht (2003)
66. Meseguer, J., Qian, X.: A logical semantics for object-oriented databases. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 89–98 (1993)
67. Minsky, M.: A framework for representing knowledge. In: Haugeland, J. (ed.) Mind Design. The MIT Press, Cambridge (1981); A longer version appeared in The Psychology of Computer Vision (1975), Republished in [13]
68. Möller, R., Haarslev, V.: Description logic systems. In: Baader, et al. (eds.) [7], ch. 8, pp. 282–305
69. Nebel, B.: Computational complexity of terminological reasoning in BACK. Artificial Intelligence 34(3), 371–383 (1988)
70. Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record 33(4), 65–70 (2004)
71. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. of Automated Reasoning 41(1), 61–98 (2008)
72. Pan, J.Z., Stamou, G.B., Stoilos, G., Thomas, E.: Expressive querying over fuzzy DL-Lite ontologies. In: Proc. of the 2007 Description Logic Workshop (DL 2007). CEUR Electronic Workshop Proceedings, vol. 250 (2007), `http://ceur-ws.org/`
73. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Publ. Co., Reading (1994)
74. Patel-Schneider, P.F., McGuinness, D.L., Brachman, R.J., Resnick, L.A., Borgida, A.: The CLASSIC knowledge representation system: Guiding principles and implementation rational. SIGART Bull. 2(3), 108–113 (1991)
75. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
76. Poggi, A., Rodriguez, M., Ruzzi, M.: Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In: Clark, K., Patel-Schneider, P.F. (eds.) Proc. of the 4th Int. Workshop on OWL: Experiences and Directions, OWLED 2008 DC (2008)
77. Racer Systems GmbH & Co. KG. Release notes for RacerPro 1.9.2 beta, `http://www.sts.tu-harburg.de/~r.f.moeller/racer/Racer-1-9-2-beta-Release-Notes/release-notes-1-9-2se8.html` (last access, July 2008)

78. Reingold, O.: Undirected connectivity in log-space. J. of the ACM 55(4) (2008)
79. Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Databases, pp. 119–140. Plenum Publ. Co., New York (1978)
80. Rodríguez-Muro, M., Calvanese, D.: An OBDA extension to the DIG 1.1 Interface (July 2008), `http://www.inf.unibz.it/~rodriguez/OBDA/dig-11-obda/`
81. Rodriguez-Muro, M., Calvanese, D.: Towards an open framework for ontology based data access with Protégé and DIG 1.1. In: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions, OWLED 2008 (2008)
82. Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing ontology based data access: A plug-in for Protégé. In: Proc. of the ICDE Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008), pp. 286–289. IEEE Computer Society Press, Los Alamitos (2008)
83. Schild, K.: A correspondence theory for terminological logics: Preliminary report. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991), pp. 466–471 (1991)
84. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
85. Sirin, E., Parsia, B.: Pellet system description. In: Proc. of the 2006 Description Logic Workshop (DL 2006). CEUR Electronic Workshop Proceedings, vol. 189 (2006), `http://ceur-ws.org/`
86. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Technical report, University of Maryland Institute for Advanced Computer Studies, UMIACS (2005)
87. Straccia, U.: Towards top-k query answering in description logics: The case of DL-lite. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 439–451. Springer, Heidelberg (2006)
88. Uschold, M., Grüninger, M.: Ontologies and semantics for seamless connectivity. SIGMOD Record 33(4), 58–64 (2004)
89. van der Meyden, R.: Logical approaches to incomplete information. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 307–356. Kluwer Academic Publishers, Dordrecht (1998)
90. Vardi, M.Y.: The complexity of relational query languages. In: Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC 1982), pp. 137–146 (1982)
91. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
92. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences 32, 183–221 (1986)
93. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Heidelberg (1999)

Author Index